[C]1 [R]Sunday 13 July, 2025

PALESTINE POLYTECHNIC UNIVERSITY
College of Information Technology and Computer Engineering
Computer Engineering Department

Whisper Hand

Submitted by:

Sara Khatib-201073

Shatha Awawdh-227044

Seba Atawneh-201024

Supervised by:

Dr. Amal Dweik

June-2025

Certification and Anti-Plagiarism Declaration

This is to declare that the graduation project produced under the supervision of Dr. Amal Dweik, having the title "Whisper Hand", was prepared by students Shatha Awawdh, Sara Khatib, and Seba Atawneh below in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering. No part of this work has been reproduced illegally (in particular: cut and paste), which can be considered as plagiarism. All referenced parts have been used to support and argue the idea and have been cited properly.

We certify that we will not commit any plagiarism, cheating, or any other academic integrity violation. I/We will be responsible and liable for any consequence if a violation of this declaration is proven.

Any use of Artificial Intelligence (AI) tools in the preparation of this project has been clearly declared and documented in the related appendix at the end of the report, in accordance with the PPU guidelines.

Date: June 1, 2025

Graduation project group's student(s):

Name: Signature:

Name: Signature:

Name: Signature:

ABSTRACT

Communication challenges faced by mute individuals in Arabic-speaking societies arise from the limited number of people who understand sign language. The ”Whisper Hand” project addresses this issue by developing an innovative system to translate Palestinian sign language into audible speech. The system utilizes a single smart glove equipped with five flex sensors to capture finger movements and an MPU-6050 motion tracking unit to detect hand orientation and acceleration. These sensors are connected to an ESP32 microcontroller, which processes the data using machine learning algorithms. To ensure real-time translation on the ESP32, the system uses a Convolutional Neural Network 1D (Conv1D) model due to its efficiency and low computational requirements. For more complex gesture recognition tasks on a computer, the system employs LSTM models. The processed data is then converted into speech through a DFPlayer module that plays pre-recorded audio responses, facilitating real-time and offline translation. The design prioritizes accuracy, ease of use, and portability, allowing mute individuals to seamlessly integrate the system into daily interactions. Major accomplishments include an extensive analysis of Palestinian sign language to define system requirements, the development of a single smart glove prototype, and the successful integration of both Conv1D and LSTM machine learning algorithms for gesture recognition. The system has been Carefully tested under various conditions to evaluate robustness and responsiveness. Essential hardware and software components have been Accurately selected to optimize performance and ensure reliable real time translation.

Key Words: Palestinian Sign Language, Smart Glove, Flex Sensors,conv1D, LSTM Algorithm.

الملخص

يواجه الأفراد البكم في المجتمعات الناطقة بالعربية تحديات كبيرة في التواصل، نتيجة قلة الأشخاص الذين يفهمون لغة الإشارة. يهدف هذا المشروع إلى حل هذه المشكلة من خلال تصميم نظام مبتكر لتحويل لغة الإشارة الفلسطينية إلى كلام مسموع. يعتمد النظام على قفاز ذكي يحتوي على مستشعرات مرنة ووحدة قياس حركة متصلة بوحدة تحكم. يعمل القفاز على التقاط حركات اليد وترجمتها باستخدام خوارزميات تعلم الآلة إلى كلمات منطوقة عبر مكبر صوت صغير، وأيضًا تم القيام بمقارنة بين خوارزميات تعلم الآلة لتحديد الأكثر دقة من بينها والأكثر ملاءمة للمشروع، وهما خوارزميتا LSTM و Conv1D . تم تصميم النظام ليكون دقيقًا وسهل الاستخدام وخفيف الوزن، مما يجعله مناسبًا للاستخدام اليومي من قبل الأفراد البكم لتسهيل تواصلهم مع المجتمع. يهدف المشروع إلى إثبات إمكانية توفير حل عملي وسهل الاستخدام يسد الفجوة في التواصل للأفراد البكم، مع تقديم توصيات لتحسين دقة التعرف على الإشارات وتوسيع مفردات النظام لتلبية احتياجات أوسع.

الكلمات المفتاحية: لغة الإشارة الفلسطينية، قفاز ذكي، أجهزة الاستشعار المرنة، خوارزمية LSTM و Conv1D .

ACKNOWLEDGEMENT

In the name of the Almighty, the most compassionate, the most merciful, who bestowed upon us strength and knowledge. As we delve into the theoretical and practical framework of this project, we are grateful for the opportunity to explore and build upon the knowledge gained throughout our studies. Our sincere appreciation goes to our project supervisor, Dr. Amal Al-Dweik, for her insightful guidance and continuous support. We also extend our gratitude to the faculty members of the College of Information Technology and Computer Engineering for their valuable contributions to our academic journey. We are also honored to have received the PALUROP scholarship in recognition of the excellence and innovation demonstrated in this project. Finally, we would like to thank our families and friends for their unwavering encouragement, which motivates us to continue striving for success in this project and beyond.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preface

This project addresses the communication barriers faced by mute individuals within Palestinian speaking communities by proposing a system that translates Palestinian Sign Language (PSL) into spoken Arabic.The solution involves the development of a smart glove capable of capturing hand and finger gestures and converting them into audible speech. The system processes sensor data using signal based recognition techniques to identify specific gestures. Two processing approaches are currently under evaluation to determine their relative effectiveness in terms of recognition accuracy and system responsiveness. The ultimate objective is to deliver a practical, reliable, and accessible communication tool that enhances the ability of mute individuals to engage more effectively in Arabic speaking environments.

1.2 Problem statement

Communication plays a vital role in ensuring equitable participation and social inclusion. However, individuals who are deaf or mute often face significant challenges in engaging with others, particularly in communities where sign language is not widely understood. The absence of effective communication tools that bridge this gap contributes to social isolation and limits access to essential services,

education, and public interaction. Although there have been efforts to bridge this divide through technology, many available solutions still fall short. They can be difficult to use, lack flexibility, or fail to respond quickly and naturally making them less practical in real world interactions. There is a growing need for a solution that can truly support natural communication. A tool that not only recognizes hand gestures accurately, but also gives immediate, spoken feedback allowing mute individuals to interact more confidently and effortlessly with those around them. To address this need, the proposed project develops a wearable system that enables real time recognition and speech output, and compares two processing approaches to evaluate their effectiveness.

1.3 Aims and objectives

In this project, we propose a system that aims to provide the following :

1. The system aims to develop technology that allows mute individuals to effectively communicate with non mute individuals by translating Palestinian Sign Language into audible speech. To achieve this aim, the following objectives need to be accomplished:

    - Design a wearable smart glove capable of translating Palestinian Sign Language gestures into audible speech, with the aim of enhancing communication between mute individuals and their communities.
    - Incorporate essential sign language terms into the system to facilitate the expression of basic daily needs.
    - Conduct an analysis of commonly used sign language terms and categorize them into groups based on shared physical features such as hand movements and finger positions. Select representative terms from each category to serve as references for system training and evaluation.
    - Ensure that the glove is user friendly, comfortably designed, and optimized for practical daily use.

2. Develop AI models to accurately translate Palestinian Sign Language terms, aiming to evaluate and compare the models based on accuracy and response time to select the most efficient algorithm for real time communication.the following objectives need to be accomplished:

    - Develop AI models that can accurately translate Palestinian sign language terms using data from a smart glove, ensuring consistency through data pre processing, and optimizing the models for fast and accurate real time performance.
    - Evaluate the performance of the models by testing their accuracy, response time, and efficiency. Analyze the results to identify the most reliable model for real time use, and document key findings, including strengths and areas for improvement.
    - Deploy the chosen model into the smart glove, integrating it into the embedded system for practical use. Test the glove in real world scenarios to evaluate usability and reliability, and gather user feedback to refine and improve performance.

3. The system is designed to improve understanding and engagement between the deaf and mute community and the general public by enabling smooth communication through an advanced one handed talking glove system. To accomplish this goal, the following objectives need to be achieved:

    - Enhance the translation of essential sign language terms to accurately convey basic daily needs.
    - Improve the functionality of the glove to support a robust and practical vocabulary for effective communication.
    - Promote greater acceptance and usage of the device to bridge communication gaps in everyday social interactions.

1.4 Requirements

The system's functional requirements are as follows:

1.4.1 Functional

1. Smart Recognition Powered by AI: The glove intelligently recognizes Palestinian Sign Language using advanced machine learning, ensuring high precision in gesture interpretation, like having a real-time translator in your hand.

2. Data Processing: Every finger movement is instantly captured and transformed from analog signals into meaningful digital patterns, making communication smooth and effortless.

3. Real-Time Performance: Our AI models are trained and optimized to respond in real time with outstanding accuracy, perfectly suited for fast, on device processing.

4. High Accuracy: We experimented with and compared multiple AI algorithms to select the most reliable model offering not just speed, but consistently accurate recognition.

5. From Gesture to Voice Instantly: Once a gesture is recognized, the system speaks it out loud using a built in speaker, delivering clear and understandable spoken Arabic in milliseconds.

6. Continuously update and improve model accuracy by incorporating user feedback and new data, ensuring the system remains accurate and adaptive over time.

1.4.2 Non functional

1. Exceptional Accuracy: Designed to understand users with confidence, the glove achieves up to 99% accuracy in recognizing Palestinian Sign Language gestures, ensuring clear, dependable communication every time.

2. Fast Response: Communication feels natural thanks to an impressively low response time of just 70 milliseconds. That means the gesture is recognized and spoken almost instantly with no noticeable delay.

3. Lightweight and Wearable: The hardware design is optimized to ensure the glove is lightweight and hand wearable, with a total weight that is suitable for on hand approximately not exceeding 70 grams to maintain comfort during prolonged single hand operation.

4. Affordability: Technology is pretty accessible. That's why the entire system is built to cost no more than $110, offering powerful AI-driven communication at a price anyone can reach.

5. Safety: The glove is designed to operate safely under all expected usage conditions. All components must function at low voltage (5V) and low current to eliminate the risk of electric shock. Electrical connections shall be insulated, and all hardware must be securely enclosed to prevent direct skin contact with active components. In the event of hardware failure, the system shall not generate heat, sparks, or any hazardous conditions, ensuring the user's hand remains protected at all times.

1.5 System Description

The primary goal of our system is to facilitate effective communication for individuals who are mute by translating Palestinian Sign Language (PSL) into audible speech. The system operates in three stages:
Stage 1: The system begins by collecting data related to hand movements, capturing the specific positions and motions required for performing gestures in Palestinian Sign Language. Each gesture is initiated and completed manually by the user.
Stage 2: The captured gesture is then analyzed by a pre trained AI model, which attempts to detect and classify the performed sign.
Stage 3: Upon successful recognition, the system processes the gesture data to determine the exact sign. A machine learning algorithm evaluates the input and identifies the corresponding meaning based on the learned patterns.
Stage 4: Finally, the recognized sign is translated into audible speech, enabling effective communication between mute individuals and others. This allows users to express complete Arabic words, enhancing interaction and supporting real time, inclusive communication.
Figure 1.1 illustrates a basic overview of the glove used in this process.

Figure 1.1: Hardware Glove Illustration

1.6 Limitations and constraints

Several limitations and constraints may impact the project's implementation and effectiveness.

1.6.1 Limitations:

1. The system may support only a limited set of Palestinian Sign Language (PSL) terms, requiring additional data to expand the term library.

2. Gesture recognition accuracy depends on how accurately the user performs each sign, as variations in movement may affect recognition.

3. The generated output, whether in audible or visual form, may lack the natural prosody, regional accent, and subtle emotional expressions typically conveyed through human speech. This absence of emotional nuances, such as enthusiasm, sadness, or urgency, can limit the expressiveness and human like quality of the communication.

4. The wearable glove, equipped with sensors and a microcontroller, relies on battery power, which may limit continuous usage time.

5. The quality of training data significantly impacts term recognition accuracy, with biases or insufficient data affecting model performance.

6. Real time translation and response speed depend on the processing capabilities of the microcontroller and the efficiency of the model, influencing user experience.

1.6.2 Constraints:

1. The ESP32 has up to 520 KB of SRAM, which may not be sufficient for complex machine learning models, large datasets, or real time processing of multiple high resolution sensor inputs. Additionally, the size of the LSTM model is significantly large compared to the available memory, making it challenging to deploy and run efficiently on the ESP32 without optimization or model compression.

2. Users need adequate time to learn to use the gloves effectively, particularly in performing gestures consistently and accurately for precise translation.

3. Integration of machine learning libraries may face compatibility issues or limitations on the ESP32 platform, potentially impacting the efficiency of gesture recognition.

1.7 Schedule

We have seven essential tasks to accomplish during this semester and the next. Figure 1.2 clearly illustrates the project schedule.

	First Semester			Second Semester			
Week	1-2	3-9	10-15	1-5	6-9	10-14	15
Selection of Project Idea							
Collecting Data and System Analysis							
System Design							
System Implementation							
System Testing and Validation							
System Operation							
Documentation							

Figure 1.2: Project Schedule Diagram

1.8 Report outline

This report is organized to provide a thorough understanding of the project. Chapter 2 presents the theoretical background and related literature. Chapter 3 discusses the system design, covering both hardware and software components along with conceptual and schematic diagrams. Chapter 4 outlines the implementation process, challenges, and testing procedures. Chapter 5 discusses the results and analysis. Finally, Chapter 6 concludes the report with insights and suggestions for future development.

Chapter 2

Theoretical Background

2.1 Preface

In this chapter, we comprehensively review the theoretical background and a survey of prior research and methodologies related to sign language translation. The chapter also explores key terminologies commonly used by individuals with speech impairments.

2.2 Theoretical Background Concepts

Arabic Sign Language (ArSL) is a primary communication medium within Arab Deaf communities. However, it faces challenges due to regional variations and the absence of a unified standard. Each region often has its unique dialect, complicating cross regional communication. According to Mahmoud Ahmad Abdel-Fattah's research [3], this diversity impedes efforts to develop a universal ArSL. The WhisperHand project specifically focuses on Palestinian Sign Language (PSL).

2.2.1 Palestinian Sign Language

Palestinian Sign Language (PSL), influenced by Jordanian Sign Language, is context dependent and relies heavily on nuanced hand movements. It faces challenges in expressing abstract or academic terms, as noted by Abdel-Fattah [4]. WhisperHand aims to bridge this gap by using wearable gloves

to convert PSL into audible speech, thereby enhancing communication effectiveness.

2.2.2 PSL Analysis

This section analyzes selected terms from PSL based on references, to determine the requirements of each sign. The analysis identifies whether the signs involve one or both hands and the specific bent fingers per hand. However, as WhisperHand currently processes terms using only one hand, entries requiring both hands are excluded in the current implementation. For clarity, the findings are summarized in Table 2.1. Which are summerized in Appendix B.

Table 2.1: Phrases and Bent Fingers Required (Right Hand Only)

Phrase	**Bent Fingers of the Right Hand**
أنا	Four
شكرا	Zero
مرحبا	Zero
اضحكتني	Five
اكل	Five
اشرب	Five
احبك	Two
موافق	Five
عمل جيد	Four

2.2.3 Machine Learning

Machine Learning (ML) serves as a core component in our system by transforming raw sensor readings into meaningful spoken language. The models are trained to recognize sign language gestures by learning patterns in time series data collected from wearable sensors specifically, five flex sensors and an MPU6050, which provides both accelerometer and gyroscope readings.
Unlike rule based systems, ML enables the model to generalize across different users and execution styles, making it more adaptable and robust. To achieve efficient and accurate gesture recognition across various deployment platforms, the project adopts two different ML approaches:

- **TensorFlow Lite**, suitable for high resource environments such as PCs and smartphones, where higher computational power allows the use of more complex models.[5]
- **Edge Impulse**, optimized for microcontrollers like the ESP32, where lightweight models must run efficiently with limited processing and memory resources.[6]

These platforms utilize two distinct deep learning architectures: Long Short Term Memory (LSTM) and 1D Convolutional Neural Networks (Conv1D). Each of these models operates differently and is suitable for different types of data processing and environments.

2.2.4 Long Short Term Memory (LSTM)

LSTM (Long Short Term Memory) is a specialized type of Recurrent Neural Network (RNN) designed for processing sequential data. LSTM networks are adept at learning long term dependencies due to their distinctive memory cell structure. This internal memory is managed through a set of gates that control the flow of information, determining what to retain, update, or discard at each time step.

LSTM networks are particularly useful in situations where the order and timing of input data are essential. They are commonly applied in gesture recognition because of their ability to learn from previous sequences and maintain context over time. Furthermore, LSTM models can be arranged in layers to create Multi layer LSTM networks, which improve the capability to learn complex patterns in temporal data.

These networks are utilized to capture the intricate dynamics of sign language gestures, taking advantage of their proficiency in recognizing temporal patterns from data collected through flex sensors and other relevant inputs which will be processed through the PC. [7]

Figure 2.1 a 3-layer LSTM model to recognize hand gestures from flex sensors and the MPU6050 sensor. The input layer takes time based sensor data, which passes through LSTM layers to learn how the gesture changes over time. The output layer gives the final predicted gesture. We used 80% of the data for training and 20% for testing and applied K-Fold Cross Validation to find the best model settings and avoid overfitting.

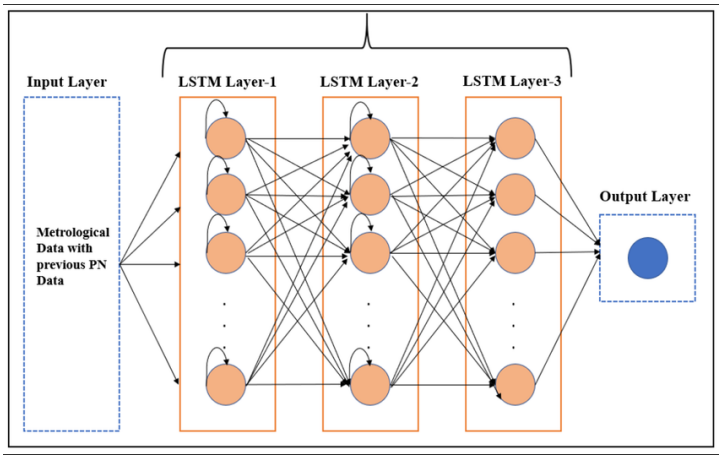

Figure 2.1: Sequential LSTM Architecture [1]

2.2.5 Conv1D

A Conv1D is a type of Convolutional Neural Network tailored for analyzing one dimensional sequences such as time series signals. It operates by applying convolutional filters over the input data to extract local patterns and features. These filters slide over the sequence to detect edges, peaks, or motion like structures within the data.

Conv1D are known for their computational efficiency and are commonly used in signal processing, speech recognition, and sensor based activity classification. They efficiently capture spatial hierarchies through convolutional layers while maintaining fast computation times, making them ideal for real time applications. Unlike recurrent models, they do not maintain temporal memory but can effectively capture short term dependencies and distinctive signal shapes. Additionally, Conv1D can combine multi-

ple convolutional layers with pooling to enhance feature abstraction and reduce dimensionality.[8] By employing both LSTM and Conv1D models, the system can leverage the strengths of each architecture depending on the hardware and performance needs. This dual approach enhances the flexibility and robustness of the WhisperHand system. Figure 2.2 shows the general architecture for the Conv1D for time series input

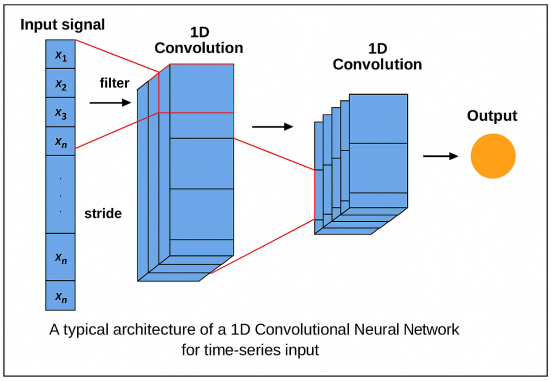

Figure 2.2: Typical architecture of Conv1D [2]

2.2.6 Finger Motion Detection

Flex sensors detect finger bending based on resistance changes, as illustrated in Figure 2.3. These variations are used as input for gesture classification [9].

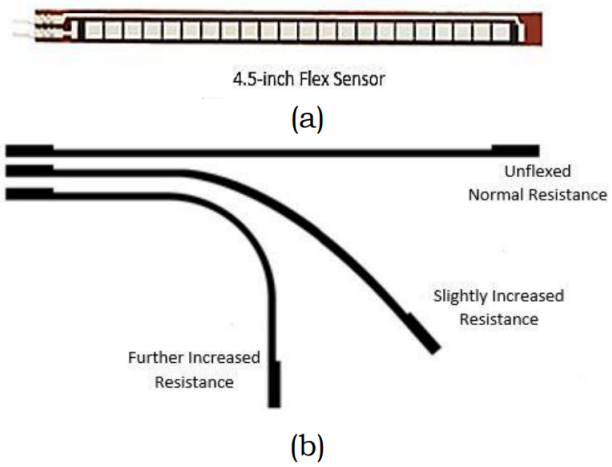

Figure 2.3: (a) Shape of a flex sensor and (b) Different flex sensor states

2.3 Literature Review

This section presents a critical and comparative analysis of existing sign language translation systems, particularly those focusing on Arabic or Palestinian contexts. The aim is to evaluate not only the effectiveness of these systems in practical terms but also their scientific approaches, such as algorithmic design, hardware choices, and validation methods. Unlike most previous projects, WhisperHand combines both a scientific research methodology through systematic experimentation on different neural networks and a practical, wearable glove.

2.3.1 Related Projects

1. **ArSL Speaking Gloves:** A two glove system that translates Arabic Sign Language into speech using flex sensors and static rule based mappings, without employing any AI or machine learning techniques. While the project claims a 98.02% success rate, this figure is not supported by any scientific evaluation, dataset transparency, or validation method making the claim highly questionable [10].

2. **Vocalizing Arabic Sign Language:** A glove based system that uses flex sensors and HMM (Hidden Markov Model) which is a statistical method used for modeling sequences by assuming that the system moves through a series of hidden states, each producing observable outputs with certain probabilities, to recognize Arabic sign language and output both text and speech. While it applies basic machine learning, the system lacks evaluation under diverse conditions and does not incorporate deep learning techniques. [11].

3. **SGTArSL:**A one hand glove system that translates Arabic Sign Language alphabets into text using predefined sensor thresholds. It relies on flex sensors, pushbuttons, and an accelerometer, and sends the recognized letter to a mobile application via Bluetooth. The system does not utilize machine learning, and recognition is based on fixed comparison with a manually coded lookup table, making it rigid and sensitive to variations between users. [12].

4. **Gloves Sign Language Translator:**A glove system that translates ASL words into English text

using Arduino and sends the result via Bluetooth to an Android app. The system is rule based and does not use machine learning, limiting its flexibility and adaptability [13].

2.3.2 Comparison With WhisperHand

Table 2.2 presents a comparative overview of previous glove based sign language translation projects in relation to WhisperHand. The comparison includes key attributes such as the type of sign language supported, number of hands required, output modality, reported accuracy, and mobile app integration. WhisperHand distinguishes itself by supporting Palestinian Sign Language using a single hand, achieving high accuracy with both Conv1D and LSTM models, and focusing on voice based output without relying on predefined mappings.

Table 2.2: Comparison of Previous Projects with WhisperHand

Project	**Sign Language**	**Hands Used**	**Output**	**Accuracy**	**Mobile App**
WhisperHand	Palestinian Terms	One Hand	Sound	TBD	Not included
SGTArSL	Arabic Letters	One Hand	Text	Not mentioned	Included
Vocalizing ArSL	Palestinian Words	One Hand	Text and sound	Nearly 100%	Not included
ArSL Speaking Gloves	Arabic Sentences	Both	Sound	98.02% (claimed)	Included
Gloves Translator	English Words	One Hand	Text	88.21%	Not included

Table 2.3 compares the artificial intelligence techniques and recognition accuracy between WhisperHand and the Vocalizing Arabic Sign Language project. While WhisperHand adopts modern deep learning approaches such as LSTM and Conv1D, the other project uses traditional HMM based modeling.

Table 2.3: Comparison between WhisperHand and Vocalizing Arabic Sign Language projects based on AI algorithms and accuracy.

Aspect	**WhisperHand**	**Vocalizing Arabic Sign Language**
AI Algorithms	LSTM, Conv1D	HMM (Hidden Markov Model)
Accuracy	TBD	almost 100%

Table 2.4 summarizes the hardware and software technologies used across different sign language glove projects. It compares the types of sensors, programming languages, microcontrollers.

Table 2.4: Sensor and Technology Comparison

Project	**Sensors**	**Language**	**Microcontroller**	**ML Used**
WhisperHand	Flex, MPU6050	Python	ESP32	Yes
SGTArSL	Flex	C++	Arduino Nano	No
Vocalizing ArSL	Flex, Accelerometer	C++	Arduino Mega 2560	Yes
ArSL Speaking Gloves	Flex, MPU6050	C++	Arduino Nano	No
Gloves Translator	Flex, MPU6050	C++	Arduino Nano	No

The review of existing glove based sign language translation systems reveals significant variation in both technical depth and practical applicability. While several projects have attempted to address the communication gap for the deaf and mute communities, many rely on static, rule based approaches without integrating modern machine learning techniques. Projects such as ArSL Speaking Gloves and SGTArSL use simple threshold logic and predefined mappings, making them rigid and highly sensitive to gesture variation between users. Although Vocalizing Arabic Sign Language adopts a machine learning approach using HMM, it still lacks deep learning capabilities and has not been evaluated under real world conditions. In contrast, WhisperHand introduces a more research driven and scalable solution by applying two advanced deep learning models LSTM and Conv1D to time series sensor data collected from flex sensors and an MPU6050. The project supports single hand input, focuses on Palestinian sign language, and achieves high recognition accuracy (up to 98.86%) without relying on handcrafted thresholds. Additionally, WhisperHand distinguishes itself by performing a systematic comparison across different model architectures and frequency domains, providing insights into the performance and responsiveness of each algorithm. This dual emphasis on scientific validation and practical usability positions WhisperHand as a unique and comprehensive solution in the field of sign language recognition, addressing the limitations observed in prior work while paving the way for future enhancements in real time, wearable communication technologies.

2.4 Summary

This chapter outlines the theoretical basis of the WhisperHand project, focusing on the structure of Arabic and Palestinian Sign Language and their contextual challenges. It explains how the system uses one hand gestures and wearable sensors (flex sensors and MPU6050) to recognize PSL phrases. The chapter highlights the use of machine learning specifically LSTM and Conv1D models trained on time series data through TensorFlow Lite and Edge Impulse. A reversed data collection setup was applied to compare model robustness. Finally, a literature review shows that unlike earlier rule based systems, WhisperHand integrates deep learning for higher accuracy and adaptability.

Chapter 3

System Design

3.1 Preface

The WhisperHand project aims to create a system that translates single-hand Palestinian Sign Language gestures into spoken words. The main objective is to improve communication for individuals who are deaf or hard of hearing by using advanced machine learning and sensor technology. The system combines hardware and software components to ensure smooth operation and an easy user experience.

This chapter offers a clear overview of the key components of the system, the design options explored, the conceptual framework, the algorithms and methods used, and the diagrams created to develop this effective and interactive solution.

3.2 System components and Design alternatives

To develop a system that accurately translates Palestinian Sign Language (PSL) into audible speech, a combination of hardware and software components is essential. These components must work in harmony to ensure reliable gesture recognition, efficient data processing, and clear speech synthesis. Below is a detailed overview of the core components and the rationale behind their selection.

3.2.1 Hardware Components and Options

Given the wide range of hardware components options available for the project, it is essential to search for and compare each component to select the most suitable ones. This section will discuss the various hardware components and their available options.

3.2.1.1 Main Controller

The system uses a microcontroller for signal processing and real-time operation, it is cost-effective compared to single-board computers, while handling real-time tasks. Table 3.1 compares three popular microcontroller and microcomputer platforms: ESP32, Arduino LilyPad, and Arduino Nano. Highlights the key features of each option, the comparison done according to [14].

Table 3.1: Comparison between Arduino LilyPad, Arduino Nano, and ESP32

Feature	**Arduino LilyPad**	**Arduino Nano**	**ESP32 WROOM**
Processor	16 MHz ATmega328P	16 MHz ATmega328P	Dual-core 240 MHz (Tensilica Xtensa LX6)
RAM	2 KB SRAM	2 KB SRAM	520 KB SRAM
Storage	32 KB Flash	32 KB Flash	Up to 16 MB Flash
Wireless Connectivity	None	None	Built-in Wi-Fi and Bluetooth
Number of Pins	14 pins	22 pins	30+ pins (depends on the model)
I/O Ports	10 digital I/O, 6 analog I/O	14 digital I/O, 8 analog I/O	34 digital I/O, 15 analog I/O
Power Supply	Battery-powered, USB	USB, Battery-powered	USB, Battery-powered, External supply
Machine Learning Support	Not feasible	Not feasible	Supports TensorFlow Lite for Microcontrollers
Cost	20$	22.49$	17.99$

The ESP32 was selected for its cost-effectiveness, compact size, and power efficiency, making it ideal for real-time gesture recognition. With Bluetooth support for seamless wireless data transmission, it integrates well with flex sensors and MPUs. While less powerful than the Raspberry Pi, the ESP32 can handle lightweight machine learning models and offers lower power consumption, making it perfect for portable, wearable systems.

3.2.1.2 Finger Movement Detection Sensors

It is essential to measure finger bending, and there are various options available for this purpose. Table 3.2 compares strain gauges and flex sensors according to [15] and [16]. Strain gauges are precise but complex, while flex sensors are simpler but less accurate.

Table 3.2: Comparison between Strain Gauge Sensors and Flex Sensors

Aspect	**Strain Gauge Sensors**	**Flex Sensors**
Features	Measures strain (deformation) precisely.	Measures bending or flexing angle.
Advantages	High accuracy; durable; suitable for precise forces.	Simple; lightweight; easily integrates with wearables.
Disadvantages	Requires complex circuitry; less flexible in wearables.	Less accurate; degrades with repeated use.
Price (per unit)	$5–20 depending on type and resistance.	$8–18 depending on size and brand.

Flex sensors are chosen for finger bending detection due to their simplicity, reliability, and seamless integration with ESP32 microcontrollers.

3.2.1.3 Sensors for Tracking Movement

The system necessitates the integration of sensors to accurately detect hand positions, and there exists a range of options available for this purpose. Table 3.3 presents a comparative analysis of various sensors utilized for movement tracking. The comparison is done according to [17] and [18].

Table 3.3: Comparison of Sensors for Movement Tracking

Sensor Type	**Features**	**Advantages**	**Disadvantages**	**Price**
MPU-6050	Combines accelerometer and gyroscope for motion tracking.	High accuracy, versatile, compact.	More expensive than basic sensors.	$12.95
ADXL335	Measures acceleration and detects tilt.	Affordable, simple, lightweight.	Limited data for precise tracking.	$14.95
L3G4200D	Measures rotational velocity.	Accurate rotation tracking.	Requires combination with other sensors for full motion tracking.	$5.95

The MPU-6050 was chosen for its precision, combining accelerometers, gyroscopes, and magnetometers to accurately track motion and orientation. Its compact size and versatility make it ideal for the our wearable system.

3.2.1.4 Power Supply

The system requires a power supply, and there are many available options to choose from. Table 3.4 compares Lithium Ion, Lithium Polymer, and Rechargeable Battery.

Table 3.4: Comparison of Battery Types

Battery Type	**Advantages**	**Disadvantages**	**Price (per unit)**
Primary Lithium Batteries	High energy output and high energy density.	Not rechargeable.	$2.00 - $5.00
Alkaline Batteries	Easily available and inexpensive.	Not rechargeable.	$0.50 - $1.00
Lithium-Ion Batteries	High energy density, suitable for compact devices. Lightweight, making it practical for portable and wearable applications. Rechargeable.	Risk of overheating.	$10.00 - $20.00

Lithium-Ion was chosen for its high energy density and rechargeability.

3.2.1.5 Audio Output

The output will be audio, so an audio output module is necessary. Many options are available. Table 3.5 compares wearable-friendly speakers for Whisper Hand, focusing on size, sound clarity, and ease of integration, the comparison is done according to [19] and [20].

Table 3.5: Comparison of Speakers for Wearable Whisper Hand Project

Speaker Type	**Advantages**	**Disadvantages**	**Price (per unit)**
Adafruit Mini Metal Speaker	Compact, lightweight, easy to integrate, suitable for wearables.	Limited volume, moderate sound clarity.	$4.95
Adafruit 0.5W Slim Speaker	Ultra-thin design, lightweight, wearable-friendly.	Lower volume, might need an amplifier for clarity.	$3.50

The Adafruit Mini Metal Speaker was chosen for its compact size, lightweight design, and suitability for wearable applications.

3.2.2 Software Components and Options

To develop our system efficiently, a combination of software platforms and machine learning algorithms is essential. The selection of each software tool is based on its compatibility with the hardware components, model performance requirements, and the system's real-time processing needs.

3.2.2.1 Machine Learning and Processing

In our project, the primary objective is to translate single-hand Palestinian Sign Language gestures into audible speech. To achieve accurate gesture recognition and efficient processing, we considered several AI algorithm options, including RNN, LSTM, and Conv1D. Table 3.6 presents a comparison between the mentioned algorithms.

Table 3.6: Comparison between RNN, LSTM, and Conv1D

Feature	**RNN**	**LSTM**	**Conv1D**
Complexity	Simpler, fewer parameters.	More complex, uses gates and cell states.	Efficient in capturing local patterns, less complex than LSTM.
Memory	Limited to short-term patterns.	Retains long-term dependencies.	Short-term dependency capturing.
Temporal Dependency	Struggles with long-term sequences.	Excels at modeling sequential data.	Captures short-term patterns effectively.
Vanishing Gradient	Prone to vanishing gradients.	Resistant, ensures stable training.	Not affected, due to convolutional nature.
Performance	Good for simple tasks.	Ideal for complex gesture recognition.	Fast and suitable for real-time processing.
Suitability for WhisperHand	Less effective for recognizing gestures.	Highly suitable for precise and robust recognition.	Suitable for detecting localized patterns.

Rationale for Choosing LSTM and Conv1D

The LSTM algorithm was selected due to its ability to learn long-term dependencies, which is essential for capturing the sequential nature of sign language gestures. On the other hand, the Conv1D

was chosen for its fast and efficient processing of short-term, localized features. The combination of these algorithms ensures high accuracy and real-time responsiveness, which are critical for the WhisperHand system.

3.2.2.2 Text to Speech Conversion

To enable speech output from text, a text-to-speech (TTS) conversion method is required. Initially, lightweight and offline TTS engines such as eSpeak or Festival were considered due to their suitability for devices like the ESP32. However, in the current implementation, we employ DFPlayer, a compact MP3 player module that directly outputs pre recorded audio files.

3.2.2.3 Data Set

The dataset was built using both TensorFlow and Edge Impulse platforms. Gesture data was collected from flex sensors and the MPU6050, processed in real time, and labeled for each gesture class. TensorFlow was used to train LSTM models, while Edge Impulse was used to train Conv1D models.

3.2.3 Data Collection Approach

Initially, the data collection process followed distinct sampling frequency setups for each model. This differentiation was due to the nature of the algorithms and their ability to process data efficiently:

- **On Edge Impulse:** The Conv1D model will be trained using low frequency data. This choice was made because Conv1D models can efficiently process data with sparse temporal resolution while still capturing essential features due to their ability to detect local patterns.

- **On TensorFlow Lite:** The LSTM model will be trained using high frequency data. This approach leverages LSTM's capability to maintain temporal dependencies and sequence information over extended periods, which is essential for accurate gesture recognition.

3.2.3.1 Reversal of Data Collection Setup

To ensure a comprehensive comparison and evaluate model robustness, we reversed the setup in the subsequent phase:

- Collected high frequency data for the Conv1D (Edge Impulse): This test aimed to evaluate whether Conv1D can handle densely sampled data and still maintain accuracy without significant performance loss.
- Collected low frequency data for LSTM (TensorFlow Lite): This adjustment aimed to assess how well the LSTM model can maintain accuracy and temporal consistency when the data is sparsely sampled.

3.2.3.2 Rationale and Impact

The rationale behind this reversal was to examine each model's adaptability to different data frequency settings. By doing so, we could measure the robustness and flexibility of both models, allowing for a fair comparison of their performance in various conditions. This dual approach ensures that the chosen model setup is not biased by data frequency, providing a balanced evaluation of both LSTM and Conv1D models within our system.

3.3 Conceptual System Description

Diagram 3.1 represents a single hand gesture recognition general block diagram designed to convert hand movements into audible speech for communication purposes.

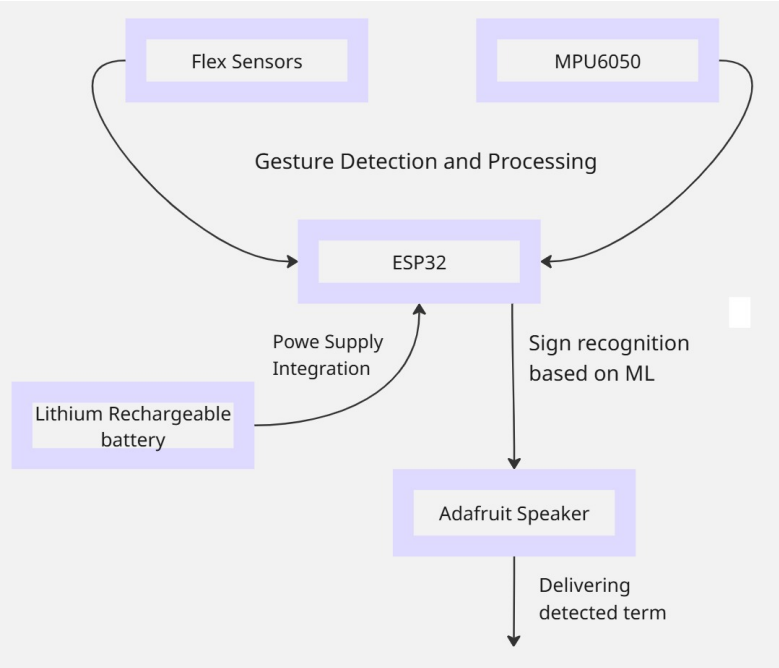

Figure 3.1: System Block Diagram

The block diagram presented a single-hand sign language translation device that employs flex sensors and an MPU6050 for the detection and processing of gestures. The ESP32 microcontroller functions as the central processing unit, responsible for collecting sensor data and executing machine learning algorithms, which are (LSTM) networks and (Conv1D), to achieve real-time gesture recognition. The system is energized by a rechargeable lithium battery, facilitating portability and sustained operation.

Upon identifying a gesture, the ESP32 activates an Adafruit speaker to audibly convey the corresponding recognized term. This configuration guarantees efficient real-time processing and dependable audio output, thereby enhancing its suitability for wearable applications.

Figure 3.2 shows how the components will be integrated with each other on each hand.

Figure 3.2: Description of The System

The conceptual diagram is shown in figure 3.3 outlines a system for translating hand gestures into audible speech to facilitate communication for mute individuals.

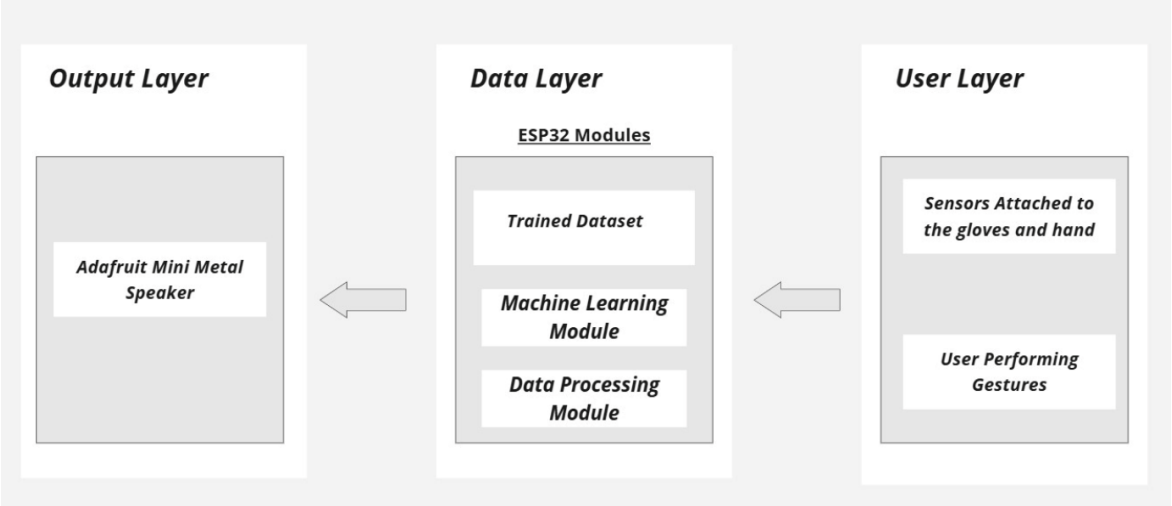

Figure 3.3: System Conceptual Diagram

The conceptual diagram features a high-level architecture that is organized into three layers. The User Layer captures gestures using sensors embedded in glove and hand attachments. These sensors

communicate with ESP32 modules to transmit motion data. The raw data is sent to the Data Layer for processing when the user performs a gesture.

The Data Layer functions as the processing hub, handling raw sensor data and recognizing gestures. It consists of a trained dataset with pre-trained machine-learning models stored on the ESP32 modules. The machine learning module analyzes the data for gesture recognition through preprocessing, feature extraction, and classification. The ESP32 processes the incoming data, identifies the gesture label, and transmits it to the Output Layer.

The Output Layer generates feedback based on the recognized gesture. Using an Adafruit Mini Metal Speaker, digital signals are converted into audio feedback. The ESP32 maps the gesture to its audio output, providing immediate spoken feedback to the user. The workflow involves capturing gestures, processing and recognizing them, and delivering feedback through the speaker.

3.4 Algorithms and Methodologies

As shown in Flowchart 3.4 the system reads data from flex sensors and MPUs to detect hand gestures. The detected data is sent to the ESP32, where an LSTM model identifies the gesture from the data set. If recognized, the result will be output as sound via the speaker; Otherwise, the system prompts the user to repeat the gesture.

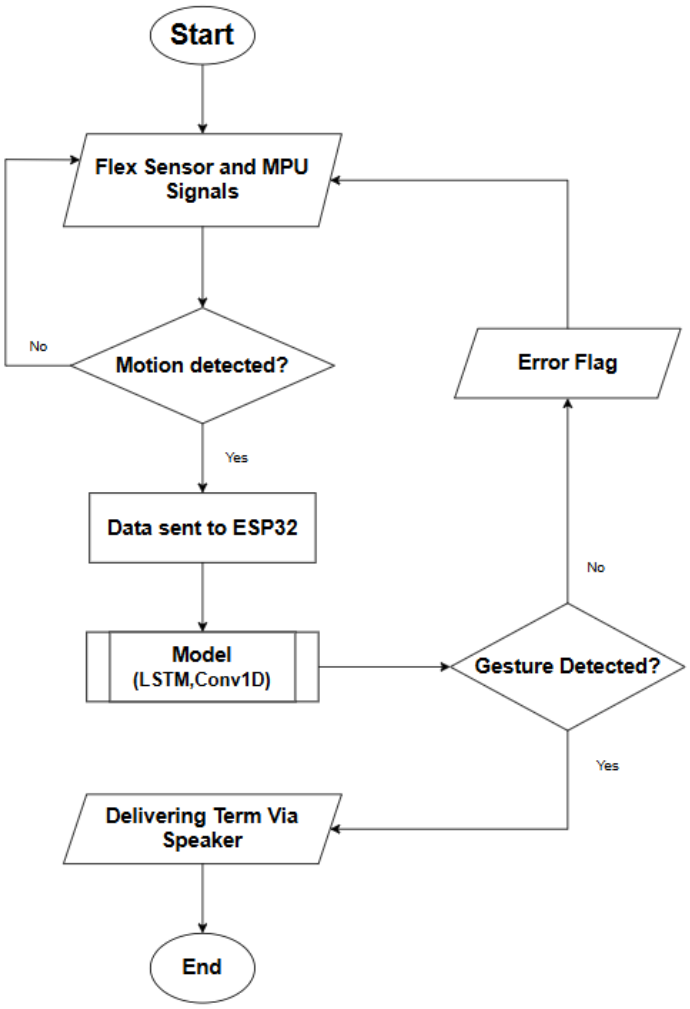

Figure 3.4: System Flowchart

To clarify, Figure 3.5 demonstrates how an LSTM model detects gestures using data from flex sensors and MPUs in a smart glove. It loads a pre-trained model, maps gesture classes to names, and processes sensor data for real-time gesture recognition, displaying results as speech.

```
Algorithm: LSTM Model for Gesture Detection

Input:    Sensor data from flex sensors and MPUs
Output:   Predicted gesture

Initialize: Load the pre-trained LSTM model
Prepare:   Create a dictionary to map gesture classes to their names (e.g., Class 3 → "Hello")

while the system is running do

    // Data Collection
    Read sensor data from the wearable glove:
        - Flex sensors: measure finger bending
        - MPUs: measure hand acceleration and rotation
    Combine all sensor readings into a single data array

    // Preprocessing
    Normalize sensor values to range [0, 1]
    Organize data into a sequence of time steps
    Reshape input to format: [samples, time steps, features]

    // Prediction
    Pass the preprocessed sequence to the LSTM model
    Get class probabilities as output

    // Identify the Gesture
    Select the class with the highest probability

    // Translate the Gesture
    Map the predicted class to its name using the dictionary

    // Output the Result
    Convert the gesture name to audio
    Play the audio through the speaker

end while
```

Figure 3.5: LSTM Model for Gesture Detection

Figure 3.6 shows a systematic approach to recognizing gestures using Conv1D. The process starts by collecting sensor data from wearable devices, such as flex sensors and motion processing units (MPUs). This raw data is preprocessed by normalizing values and organizing them into sequences for input into a convolutional neural network (CNN). The preprocessed data then passes through convolutional layers that detect patterns and features, followed by pooling layers that reduce dimensionality while preserving important characteristics. The flattened output is classified through fully connected layers. Finally, the predicted gesture is mapped to its name and conveyed as audible speech, enabling efficient real-time gesture translation.

```
Algorithm: Conv1D Model for Gesture Detection

Input:    Sensor data from flex sensors and MPUs
Output:   Predicted gesture

Initialize: Load the pre-trained Conv1D model
Prepare:   Create a dictionary to map gesture classes to names (e.g., Class 3 → "Hello")

while the system is running do

    // Data Collection
    Read sensor data from the wearable glove:
        - Flex sensors: measure finger bending
        - MPUs: measure hand acceleration and rotation
    Combine all readings into a single data array

    // Preprocessing
    Normalize sensor values to range [0, 1]
    Organize data into a sequence of time steps
    Reshape input to format: [samples, time steps, features]

    // Feature Extraction using Conv1D
    Pass the sequence through convolutional layers
    Apply filters to extract local patterns in time-series data
    Perform pooling to reduce dimensionality and retain essential features

    // Classification
    Flatten the feature maps
    Pass them through fully connected layers
    Obtain class probabilities as output

    // Identify the Gesture
    Select the class with the highest probability

    // Translate the Gesture
    Map the predicted class to its name using the dictionary

    // Output the Result
    Convert the gesture name to audio
    Play the audio through the speaker

end while
```

Figure 3.6: Conv1D Model for Gesture Detection

3.5 Schematic Diagrams

In this section, we will provide a detailed explanation of the schematic diagram, breaking it down part by part for clarity. Figure 3.7 shows the connection of flex sensors to the ESP32 controller. ensuring consistent functionality throughout the system.

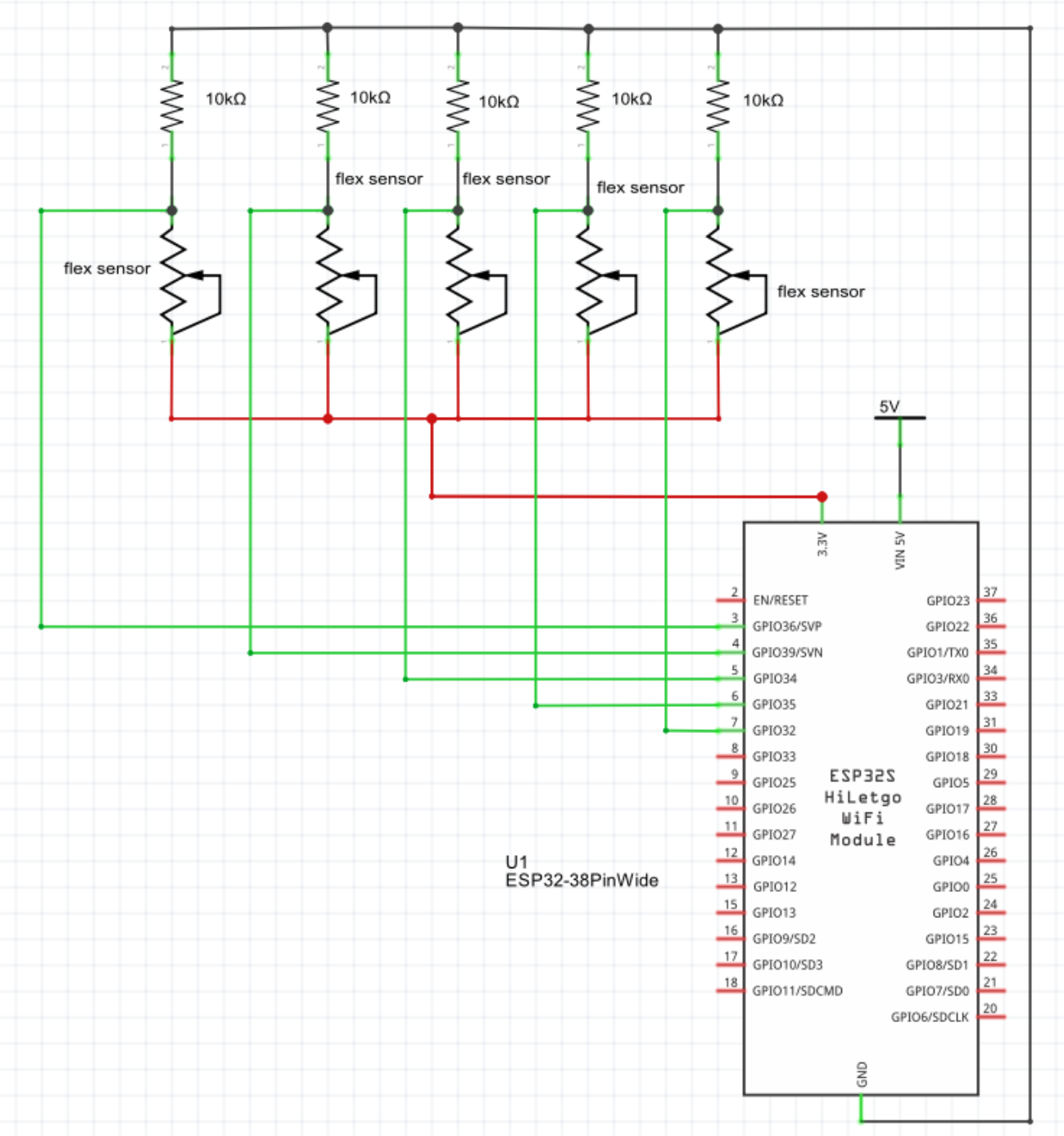

Figure 3.7: Flex Sensors Connection With ESP32

Diagram 3.8 shows an ESP32 connected to an MPU6050 sensor via I2C (GPIO21 for SDA and GPIO22 for SCL). The ESP32 is powered by 5V, with a regulator providing 3.3V. It's used for motion tracking or gesture recognition.

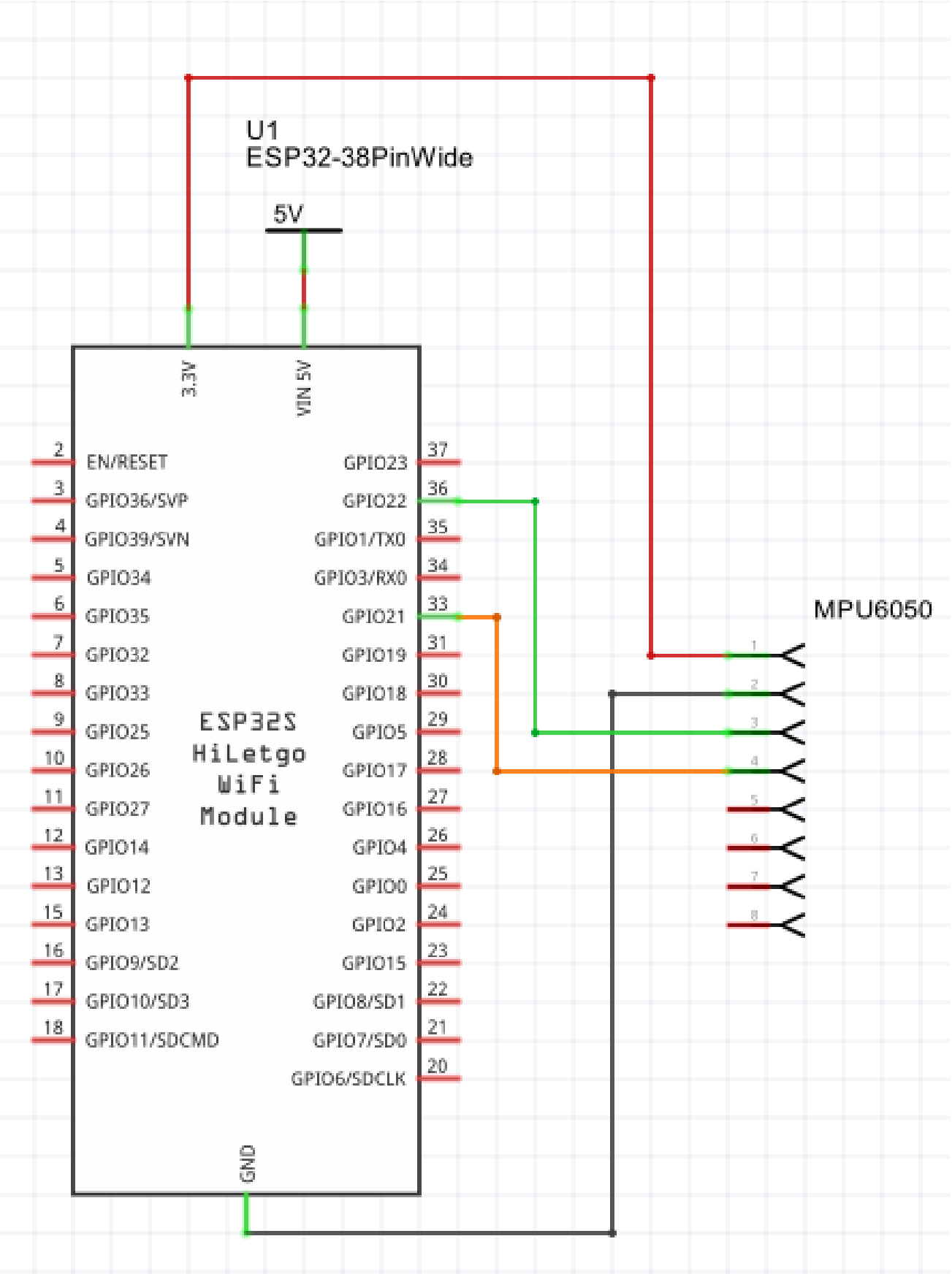

Figure 3.8: MPU6050 Connection With ESP32

Diagram 3.9 llustrates the connection between the ESP32 module, the DFPlayer Mini, and a speaker. The ESP32 sends control signals to the DFPlayer, which directly outputs the audio signal to the speaker. This configuration eliminates the need for an external amplifier, as the DFPlayer itself amplifies the audio signal to produce sound.

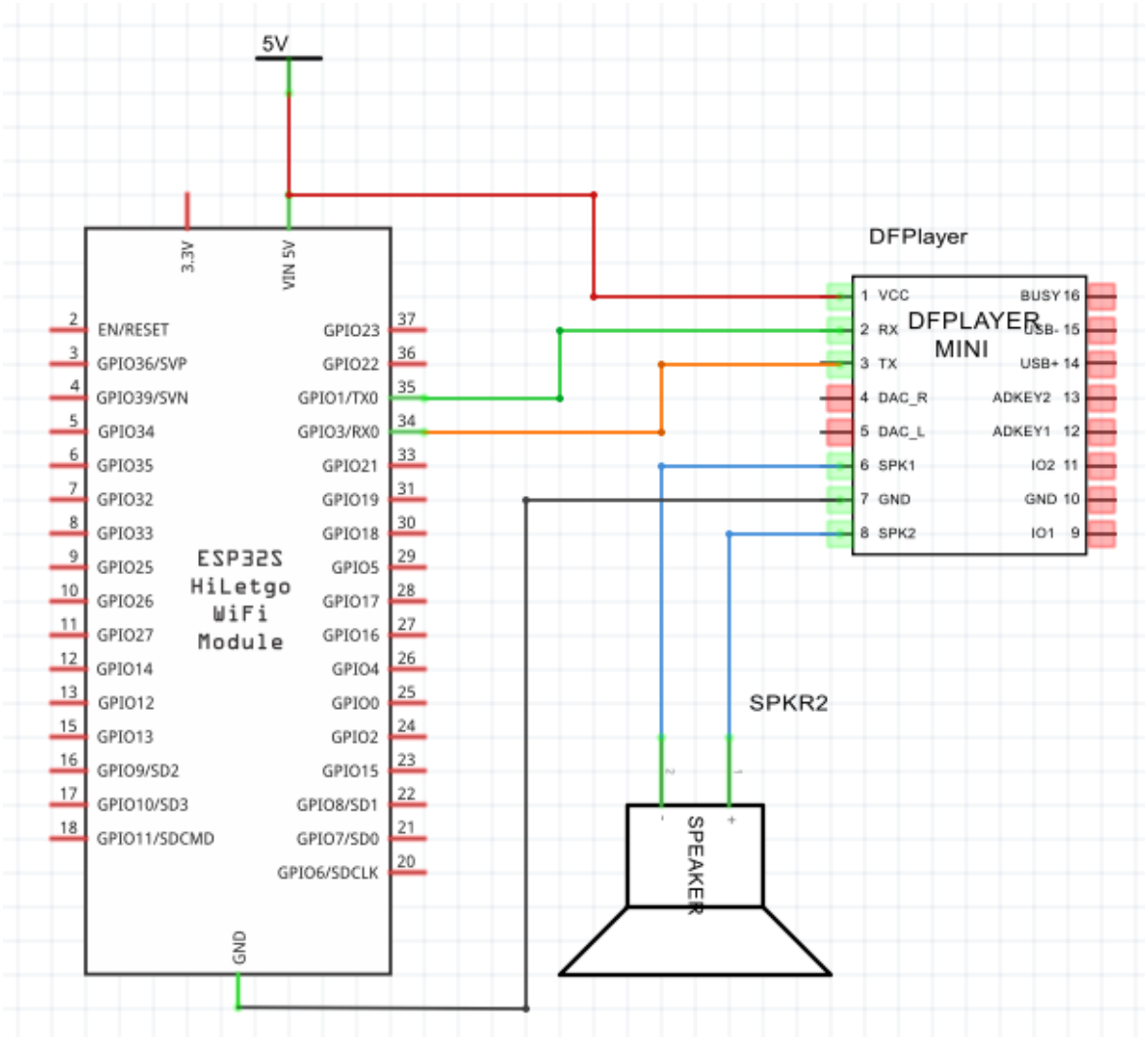

Figure 3.9: Speaker and DFPlayer Mini Connection

The system's whole connection on hand is shown in Figure 3.10

•

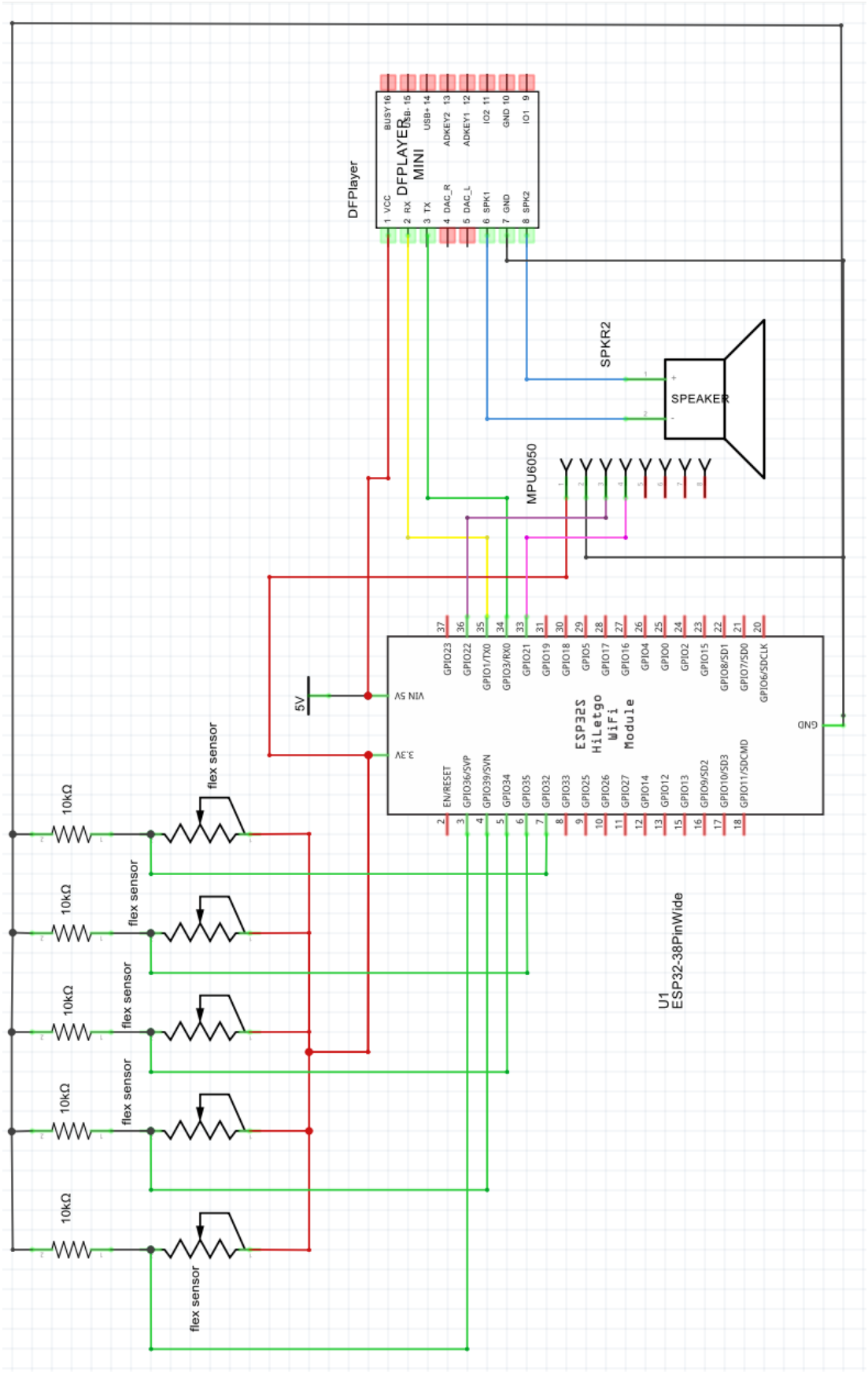

Figure 3.10: Full Schematic Diagram

3.6 Summary

WhisperHand is an innovative system designed to translate single-hand Palestinian Sign Language (PSL) gestures into audible speech. The system utilizes an ESP32 microcontroller, equipped with flex sensors to detect finger movements and an MPU6050 for capturing hand orientation and motion. To achieve accurate gesture recognition, the system integrates both LSTM and Conv1D algorithms, leveraging their strengths in temporal pattern recognition and fast inference. The processed gestures are converted into voice output using an Adafruit speaker. The design prioritizes real-time processing, portability, and reliable performance, making it a practical solution for enhancing communication for individuals who are deaf or hard of hearing.

Chapter 4

Implementation

4.1 Preface

This chapter summarizes the implementation of our system, highlighting challenges, solutions, and testing procedures to ensure accuracy. Key issues included hardware and software integration, data inconsistencies, and performance bottlenecks, illustrated with visual documentation. We detail validation strategies like unit, integration, and system testing to assess performance in real-world scenarios. The chapter concludes with insights from the implementation and testing phases, demonstrating the feasibility of our system and strategies for building gesture recognition models on resource-constrained devices.

4.2 Data Collection

In this section, the data collection phase of the WhisperHand project will be explained in detail, covering the preparation, acquisition, and storage of data.

4.2.1 Data Collection Preparation

Before initiating data collection, the hardware and software components were carefully set up to ensure accurate and reliable signal capture. The setup process included hardware assembly, sensor calibration,

and microcontroller configuration.

4.2.1.1 Hardware Setup

The glove used in this project was equipped with five flex sensors to measure finger bending and one MPU6050 sensor to track hand movement. The sensors were securely attached to the glove to minimize interference and ensure consistent readings.

4.2.1.2 Sensor Calibration

The flex sensors were calibrated to ensure accurate measurement of bending angles. The MPU6050 sensor was configured with an accelerometer range of ±8g (where g represents the acceleration due to gravity) and a gyroscope range of ±500°/s, allowing it to effectively capture fast and dynamic hand movements.

4.2.1.3 ESP32 Configuration

The ESP32 microcontroller was programmed to acquire data from the flex sensors via analog GPIO pins and from the MPU6050 using the I2C protocol. Serial communication was configured to transmit the captured data to a connected PC for analysis and storage.

4.2.2 Data Acquisition Process

The data acquisition process in the WhisperHand project was systematically designed to capture hand gestures with high temporal fidelity. After analyzing the structure of gestures in Palestinian Sign Language, it was observed that most individual gestures typically last no longer than 2 seconds. To ensure the capture of fine grained motion details, the sampling frequency was initially set to 50 Hz. However, testing revealed that this rate did not provide enough temporal resolution for accurate gesture recognition. Therefore, a higher frequency of 100 Hz was tested. While it provided more detail, it led to inconsistent classification results due to timing issues either capturing the gesture too early or too late. This misalignment caused overlap between gesture segments, which confused the model during

classification and reduced overall accuracy. To address this, the sampling rate was increased to 250 Hz, which offered better alignment with gesture dynamics and improved classification performance. This choice is supported by Vásconez et al [21], who demonstrated that higher sampling frequencies (up to 1 kHz) significantly enhance the accuracy and responsiveness of gesture recognition systems.

4.2.2.1 General Data Acquisition Workflow

The data acquisition process involved continuously reading sensor values and transmitting the data to a connected PC. The ESP32 microcontroller handled the acquisition from both flex sensors and the MPU6050 in a synchronized loop to maintain consistency. The acquired data was formatted into structured packets and sent via serial communication for storage.

4.2.2.2 Sampling Configuration

To assess the effect of sampling rate on signal accuracy and model performance, data was collected at two different frequencies, each with a unique configuration method:

1. **50Hz :** This frequency was configured using delay functions in the ESP32 firmware.
    - At 50 Hz, data was collected via Edge Impulse with sample durations ranging from 2 to 3 seconds, due to the natural timing of sign language gestures.
2. **250Hz:** This frequency was configured using the `micros()` function for higher precision.
    - Each sample lasted 1.5 seconds, resulting in 375 readings per feature. This duration was chosen to provide adequate temporal resolution for capturing the complete motion characteristics of each gesture.

4.2.2.3 Repetitions per Class and Impact

To improve model accuracy and ensure generalization across different users, the data collection process initially began with approximately 30 samples per word. However, early evaluations revealed that this

number was insufficient for achieving acceptable performance. As a result, the number of samples per word was gradually increased—first to 100, then 150, and finally to 300 samples.

Importantly, data was collected from approximately 15 different individuals, each performing the same set of gestures multiple times. This diversity in participants helped introduce natural variations in gesture execution styles, hand sizes, and motion dynamics. The inclusion of multiple users significantly enhanced the robustness of the dataset and allowed the model to learn gesture patterns more effectively, resulting in noticeable improvements in classification accuracy at each stage of repetition increase.

4.2.3 Sample Structure

Each collected sample consisted of 11 features, representing the full hand motion and finger flexion. These features included:

1. 3-axis accelerometer data (X, Y, Z)
2. 3-axis gyroscope data (X, Y, Z)
3. 5 flex sensor readings (one for each finger)

The number of readings per sample varied based on the sampling frequency and duration:

At 50Hz Sampling Frequency:

- Each sample included approximately 44 readings per second.
- The recording duration ranged from 2 to 3 seconds, generating around 88 to 132 rows per sample.
- These samples were stored in JSON format, which is a lightweight, hierarchical (key-value based) data representation suitable for programmatic manipulation and integration with software tools.

At 250Hz Sampling Frequency:

- Each sample included 375 readings captured over 1.5 seconds, providing a much denser and higher resolution time series.

- These samples were stored in Excel format, which organizes data in a tabular structure, and it was chosen in this context to facilitate offline analysis, comparison, and cross-validation during the high-frequency model training phase.

4.2.4 Data Logging and Storage

To maintain flexibility in data handling, sensor readings captured by the ESP32 were transmitted via serial communication and stored based on the chosen sampling configuration:

- **50Hz:** Data was streamed directly to Edge Impulse for cloud-based storage and processing.
- **250Hz:** Data was collected locally using a Python script, with each sample saved in a dedicated Excel file.
- **Data Organization:** Samples were organized into separate folders per gesture class, with consistent naming that included the gesture label and timestamp for easy traceability.

The table 4.1 below summarizes the key differences in sampling configuration, data collection methods, and storage approaches used in the WhisperHand project across different frequencies:

Table 4.1: Summary of Sampling Configurations

Sampling Frequency	**Configuration Method**	**Collection Tool**	**Duration**	**Readings**	**Format**	**Storage**
50Hz	delay	Edge Impulse	2–3 sec	$\sim$44/sec $\times$ duration	JSON	Edge Impulse
250Hz	`micros()`	Python Script	1.5 sec	375	Excel	Local folders (by class)

4.2.5 Data Collection Challenges

During the data collection process, several challenges were encountered that affected data quality and system stability. These challenges are categorized into hardware integration issues, data acquisition issues, and real-time data collection challenges.

4.2.5.1 Hardware Integration Issues

One of the most common challenges was related to hardware integration:

1. **Unstable Soldering:** During data collection using Edge Impulse, an issue occurred where the system displayed the error shown in Figure 4.1. This problem was caused by unstable soldering of the sensor connections, leading to intermittent disconnections between the sensors and the ESP32 microcontroller. As a result, the data acquisition process was interrupted, leading to empty payloads.

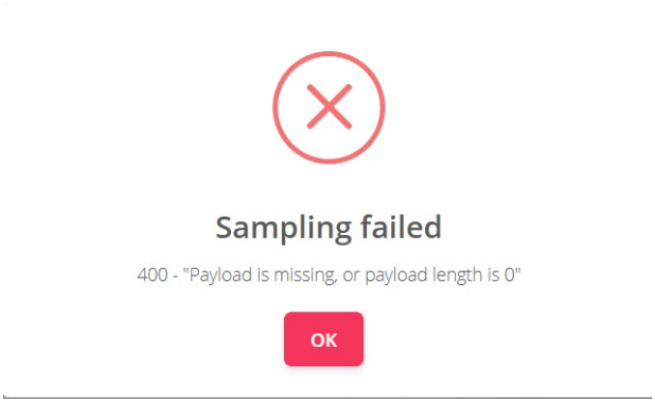

Figure 4.1: Sampling Error

   - **Solution:** Reinforcing the soldering points ensured stable connections and reliable data capture.

2. **Wiring Issues:** Figure 4.2 shows a picture illustrating how improperly cut soldered wires can lead to connection issues and data collection errors. To address this problem, the soldering points were reinforced, and the wiring was secured to prevent disconnections during movement.

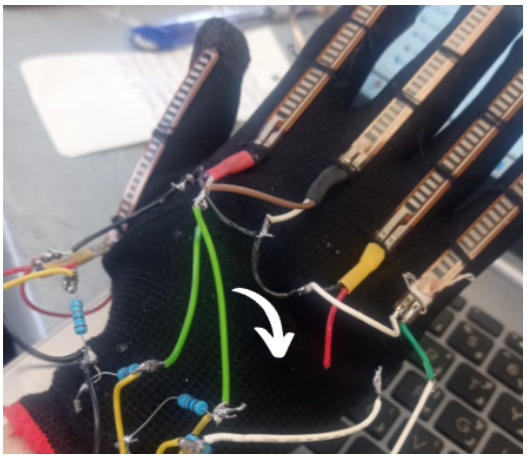

Figure 4.2: Loose In Wires Connection

4.2.5.2 Data Acquisition Issues

Several issues arose during the data acquisition phase:

1. **Zero Value Readings:** Figure 4.3 shows raw data captured from the Serial Monitor, highlighting a series of zero values in the sensor readings. This issue typically arises due to loose or unstable connections between the sensors and the microcontroller, often caused by poor soldering or mechanical stress on the wires.

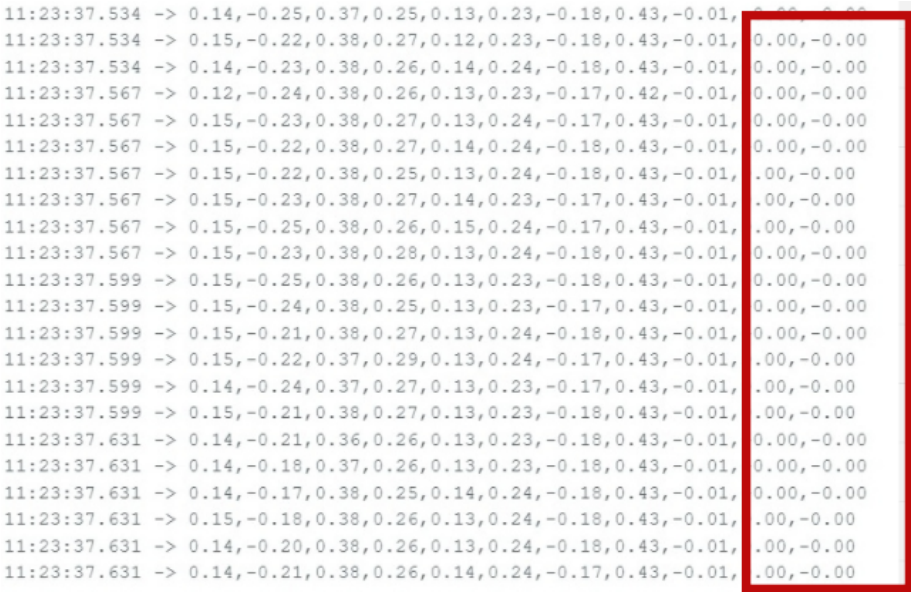

Figure 4.3: Serial Monitor Results

- **Solution:** Reinforcing the soldering points and securing the wiring minimized disconnections and prevented zero value occurrences.

2. **Sampling Scales Imbalance:** Figure 4.4 shows how low sampling frequency caused smaller sensor readings to be neglected. This imbalance occurred because rapid changes in the sensor data overshadowed subtle movements.

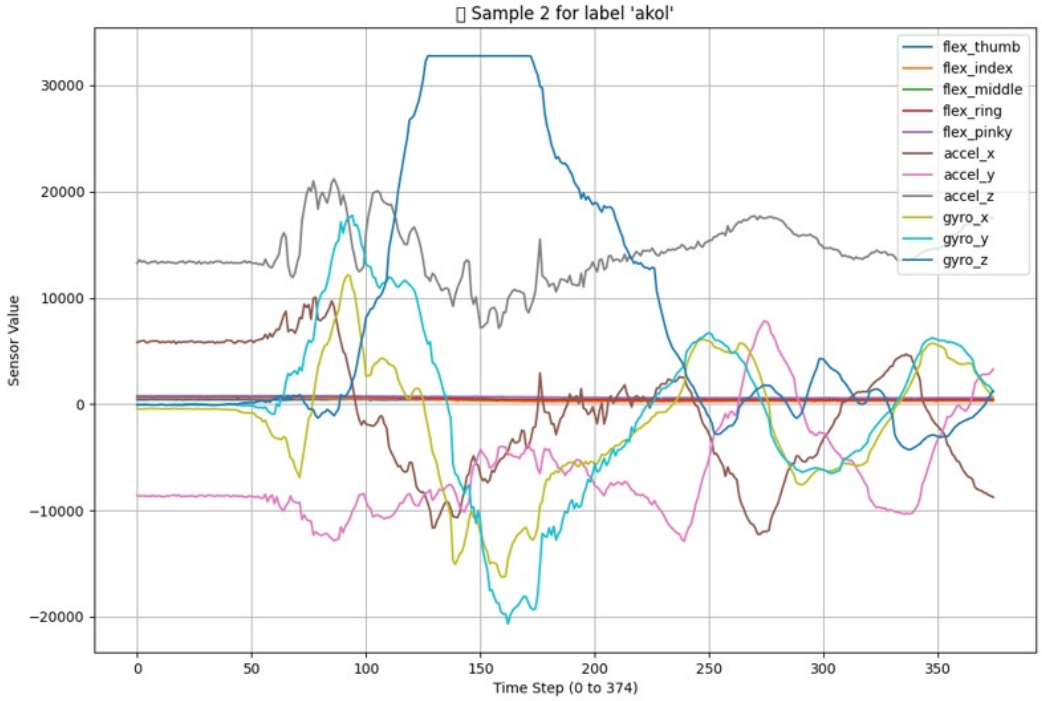

Figure 4.4: Flex Sensor and MPU6050 Readings Imbalance

- **Solution:** Adjusting the sampling frequency and applying signal normalization helped balance the captured movements.

3. **Scale Inconsistency between MPU and Flex Sensors:** As shown in Figure 4.5, MPU readings were significantly smaller compared to flex sensor values. This issue was due to incorrect scale configuration for the MPU sensors.

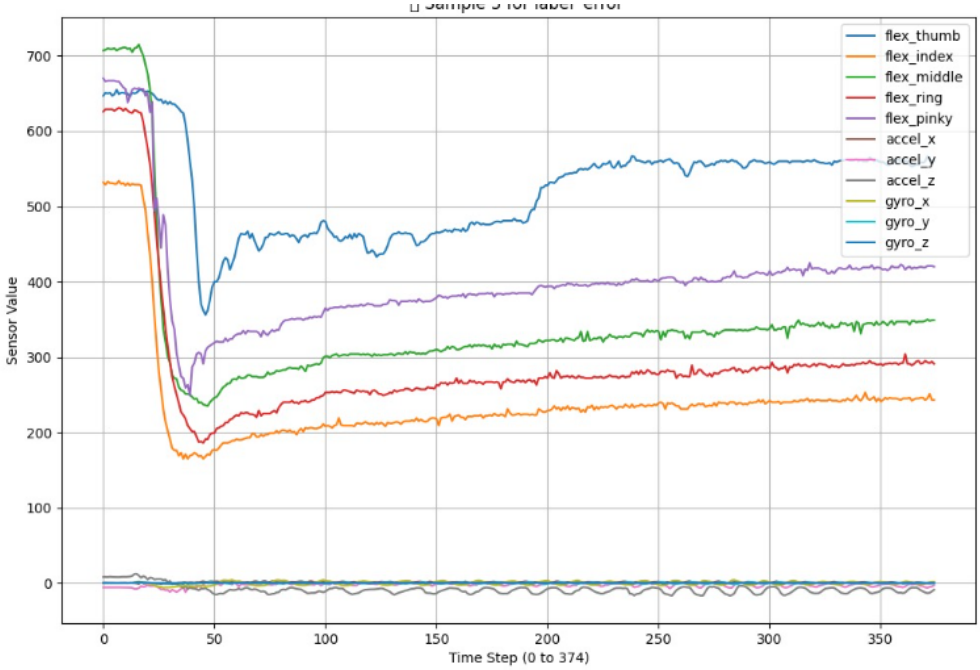

Figure 4.5: Reading Scale Issue

- **Solution:** Adjusting the MPU scaling factor to amplify the readings ensured balanced input for model training.

4.3 Model Building

This section describes the entire process of building the machine learning models, including data preprocessing, model selection, model architecture, training, and evaluation.

4.3.1 Data Preprocessing

Before training the model, data preprocessing was conducted to ensure data quality and compatibility with the chosen model architectures. Figure 4.6 (a) and (b) compare the data preparation pipelines for the LSTM and Conv1D models, respectively. In the LSTM pipeline, data is trimmed or padded, loaded, processed, split into training and testing sets, encoded as one-hot, and saved. The Conv1D pipeline follows a similar structure, involving data loading, processing, splitting, encoding, and saving. Both pipelines ensure that the data is properly formatted and consistent before being used for training.

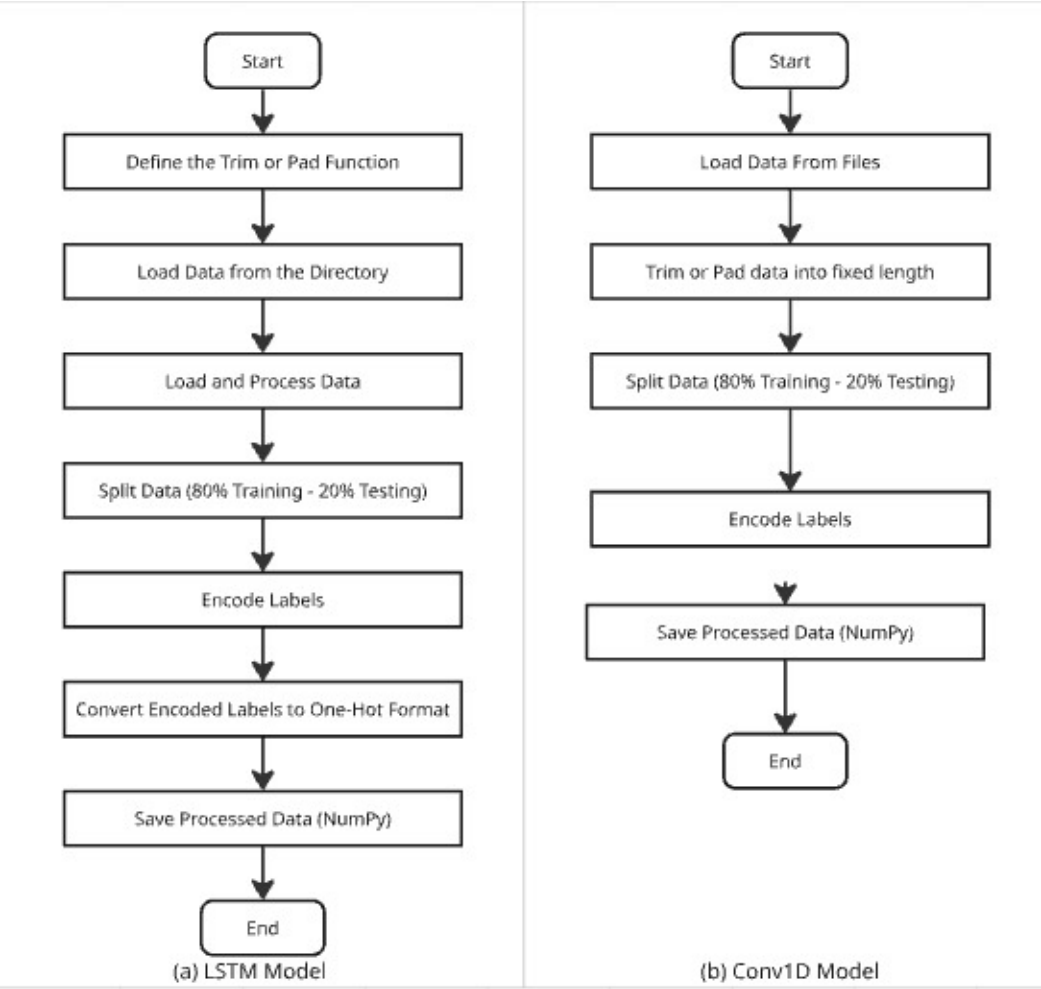

Figure 4.6: Data preparation for LSTM and Conv1D model

4.3.1.1 Data Normalization

Normalization is the process of scaling sensor values to a fixed range, typically between -1 and 1, to ensure consistency and improve model performance. The normalization formula used is shown in Equation 4.1:

$$\text{normalized\_value} = 2 \times \left(\frac{\text{value} - \text{min}}{\text{max} - \text{min}} \right) - 1 \tag{4.1}$$

This equation transforms raw sensor readings into a uniform range, which helps enhance the stability and efficiency of neural network training.

4.3.2 Model Architecture

Here, the architecture of each of the models used will be explained.

4.3.2.1 LSTM Model Architecture

Figure 4.7 shows the LSTM model architecture used for gesture recognition. It starts with an input layer (375, 11), followed by three stacked LSTM layers with 512, 256, and 128 units. Each LSTM layer is followed by dropout and batch normalization to reduce overfitting and stabilize training. An attention layer is then used and concatenated with the LSTM output. A GRU layer with 64 units follows, along with layer normalization. Finally, two dense layers are used: one with 128 units for feature extraction and another with 10 units for classification. This structure effectively captures temporal patterns while maintaining generalization.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 375, 11)	0	-
lstm (LSTM)	(None, 375, 512)	1,073,152	input_layer[0][0]
dropout (Dropout)	(None, 375, 512)	0	lstm[0][0]
batch_normalization (BatchNormalization)	(None, 375, 512)	2,048	dropout[0][0]
lstm_1 (LSTM)	(None, 375, 256)	787,456	batch_normalization[0][0]
dropout_1 (Dropout)	(None, 375, 256)	0	lstm_1[0][0]
batch_normalization_1 (BatchNormalization)	(None, 375, 256)	1,024	dropout_1[0][0]
lstm_2 (LSTM)	(None, 375, 128)	197,120	batch_normalization_1[0][…
attention (Attention)	(None, 375, 128)	0	lstm_2[0][0], lstm_2[0][0]
concatenate (Concatenate)	(None, 375, 256)	0	lstm_2[0][0], attention[0][0]
batch_normalization_2 (BatchNormalization)	(None, 375, 256)	1,024	concatenate[0][0]
gru (GRU)	(None, 64)	61,824	batch_normalization_2[0][…
layer_normalization (LayerNormalization)	(None, 64)	128	gru[0][0]
dense (Dense)	(None, 128)	8,320	layer_normalization[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 10)	1,290	dropout_2[0][0]

Figure 4.7: LSTM Model Architecture

4.3.2.2 Conv1D Model Architecture

The Conv1D model architecture shown in figure 4.8 consists of a reshape layer to structure the input, followed by a Conv1D layer with 64 filters to capture spatial patterns, and batch normalization for training stability. MaxPooling1D reduces dimensionality, followed by another Conv1D layer with 32 filters. The output is then flattened, passed through a dense layer with 64 neurons for feature extraction, and finalized with a dropout layer to prevent overfitting. The final dense layer with 10 neurons outputs gesture classifications, making the model efficient for time-series recognition.

Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 375, 11)	0
conv1d (Conv1D)	(None, 375, 64)	2,176
batch_normalization (BatchNormalization)	(None, 375, 64)	256
max_pooling1d (MaxPooling1D)	(None, 187, 64)	0
conv1d_1 (Conv1D)	(None, 187, 32)	6,176
flatten (Flatten)	(None, 5984)	0
dense (Dense)	(None, 64)	383,040
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Figure 4.8: Conv1D Model Architecture

4.3.3 Training Process

The training process consists of several steps, as illustrated in the flowchart 4.9, which outlines the procedure for both LSTM and Conv1D models. It begins with Data Loading from preprocessed files, followed by Model Building using appropriate layers. Next, the Loss Function and Optimizer are selected (typically Categorical Crossentropy and Adam). The training is then conducted using the fit() function, while Hyperparameter Tuning is performed through callbacks such as EarlyStopping. Finally, the model undergoes evaluation and is saved if it achieves the optimal performance.

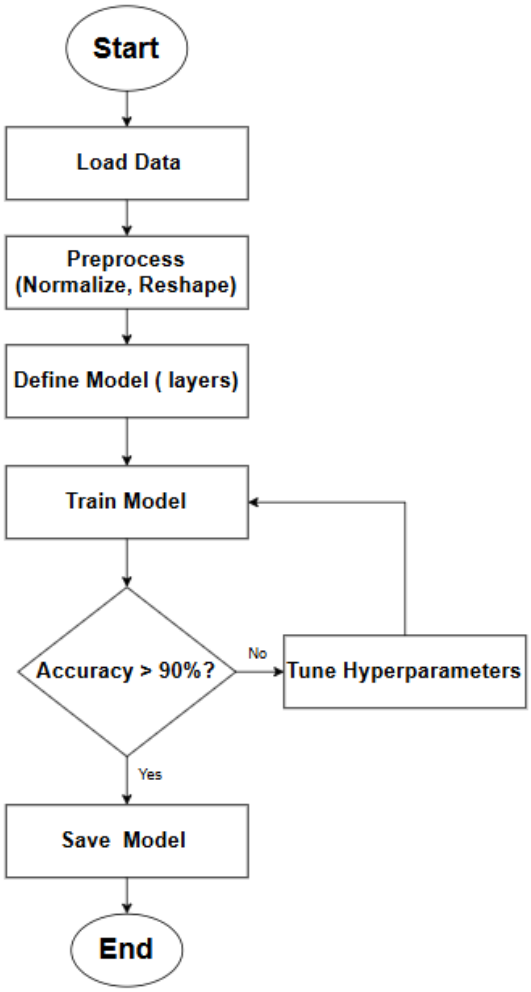

Figure 4.9: Training Flow Chart

4.3.3.1 Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing hyperparameters that significantly impact the model's performance. In both the **LSTM** and **Conv1D** models, tuning is achieved through the use of **Callbacks** that dynamically adjust training parameters to enhance accuracy and generalization [22]. In this project, hyperparameter tuning is implemented using specific techniques as follows:

1. **EarlyStopping:** Used to stop training if the model's accuracy on the validation set does not improve for a specified number of epochs (patience). Figure 4.10 shows an example of EarlyStopping.

```
early_stop = EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True, verbose=1)
```

Figure 4.10: EarlyStopping Code Example

This approach prevents overfitting by stopping early when the model's performance stagnates.

2. **ReduceLROnPlateau:** Used to reduce the learning rate when the model's performance plateaus, helping it converge more effectively. Figure 4.11 shows an example of ReduceLROnPlateau.

```
lr_reduce = ReduceLROnPlateau(monitor='val_loss', patience=5, factor=0.5, verbose=1)
```

Figure 4.11: ReduceLROnPlateau Code Example

This method allows the model to fine-tune its learning rate when improvements become minimal.

3. **ModelCheckpoint:** Used to save the model weights when a better performance is achieved. Figure 4.12 shows an example of ModelCheckpoint.

```
model_checkpoint = ModelCheckpoint(
    'best_lstm_44hz_model.h5',
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1
)
```

Figure 4.12: ModelCheckpoint Code Example

This technique ensures that the best model configuration is preserved during training.

4.3.3.2 Loss Function and Optimizer

The loss function measures how closely the model's predictions match the actual values, while the optimizer updates the model's weights to minimize the loss. In this project, the loss function and optimizer are implemented as follows:

1. **Loss Function:**The Categorical Crossentropy loss function is used because the problem involves multi-class classification. It calculates the difference between the predicted probabilities and the actual class distribution. Figure 4.13 below shows an example of Loss Function. This choice ensures that the model effectively learns the differences between multiple gesture classes.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 4.13: Loss Function Code Example

2. **Optimizer:** The Adam optimizer is chosen for its ability to adapt the learning rate during training, which results in faster convergence and improved stability.

4.3.4 Model Deployment

The model deployment phase involves preparing the trained model for practical use in the target environment. After completing training and evaluation, the models are exported to formats that support real-time usage. In this project, the deployment strategy differs between the LSTM and Conv1D models due to hardware limitations and application requirements.

Conv1D Model Deployment:

The Conv1D model is specifically optimized for deployment on embedded devices like the ESP32. Due to its lightweight architecture, it can be efficiently compressed and deployed. To reduce the model size while maintaining performance, the model is compressed using Edge Impulse with the TensorFlow Lite library. This process significantly reduces the memory footprint, making the model suitable for real-time gesture recognition on the ESP32.

LSTM Model Deployment:

The LSTM model is trained, tested, and deployed for real-time usage on a computer. The reason for using a computer rather than an embedded device like ESP32 is that LSTM models are computationally intensive and require more memory than the ESP32 can handle. After training, the model is exported in

the H5 format, allowing for efficient loading and inference on desktop systems. This setup is suitable for real-time gesture recognition on PCs where computational resources are sufficient.

4.3.5 Algorithmic Challenges and Model Building Issues

During the training phase, we encountered several challenges related to model building, data handling, and hardware limitations. These challenges are categorized as follows:

1. **Training Process Challenges:**
    - High Dependency on Experimentation:
        - The training process required extensive experimentation to fine-tune hyperparameters and model configurations.
        - This trial-and-error approach often resulted in significant time consumption, as minor changes in the architecture or hyperparameters required multiple retraining sessions.
    - Lack of Established Guidelines: Due to the unique nature of the data and the hybrid approach between LSTM and Conv1D, there were no direct references or guidelines for the optimal configuration, necessitating multiple experiments.
    - Optimization Complexity: Balancing between accuracy and model size was challenging, particularly for deployment on embedded systems.
2. **Data Handling Challenges:**
    - Different Handling Methods Across Algorithms:
        - The LSTM model requires raw, time-series data to capture temporal dependencies, while the Conv1D model benefits from spectral features obtained using FFT.
        - Handling these two different data formats within the same pipeline was complex and required separate preprocessing strategies.
    - Data Inconsistency: Variations in the data collected from different gestures sometimes led to overlapping patterns, making classification more difficult.

- Normalization Requirements: Ensuring that both raw data and spectral data were normalized correctly was essential to maintain consistency during training.

3. **Hardware Limitations:**

   - Limited Hardware Capacity:
     - The ESP32 device could not handle the LSTM model due to its high memory and processing requirements.
     - This limitation forced us to choose the Conv1D model, which was more lightweight and could be compressed efficiently using TensorFlow Lite through Edge Impulse.
   - Deployment Constraints: Even with compressed models, real-time performance was sometimes affected due to the limited processing power of the embedded device.
   - Real-Time Inference Challenges: Achieving low-latency inference on ESP32 required optimizing both the model architecture and the data preprocessing pipeline.

4.4 Summary

This chapter presents the implementation of the WhisperHand system, covering hardware setup, data collection, model training, and deployment. A glove equipped with flex sensors and a motion sensor was used to capture gesture data via an ESP32 microcontroller at different sampling frequencies (50Hz and 250Hz). After preprocessing, normalization, and feature extraction, machine learning models were trained. The system faced several challenges, including wiring issues, data inconsistencies, and hardware limitations, which were addressed through software adjustments and hardware improvements. The chapter demonstrates the system's effectiveness in real-time gesture recognition on resource-constrained platforms.

Chapter 5

Testing and Validation

5.1 Preface

The primary objective of this chapter is to clarify the testing and validation processes employed in our project, which seeks to translate Arabic sign language gestures into spoken language through machine learning models. Testing and validation are essential components in the development of any system, as they ensure the accuracy, reliability, and responsiveness of the model. This chapter comprehensively outlines the testing methodology adopted, the validation metrics utilized, and the results of our analysis. Additionally, it addresses the challenges encountered throughout the process, offering insights into the complexities of achieving effective sign language translation.

5.2 Testing Methodology

This section outlines the testing approaches used to verify the system's functionality and performance. It includes the types of tests conducted, tools used, and steps taken to ensure the system works correctly and meets the project requirements.

5.2.1 Data Splitting

The dataset was split into two parts:

1. Training Set (80%)

2. Testing Set (20%)

5.2.2 Cross-Validation

Cross-validation is a crucial technique used to evaluate the performance of machine learning models, ensuring that they generalize well to new, unseen data. In the WhisperHand project, cross-validation was employed to evaluate the robustness and stability of both the LSTM and Conv1D models used for palestinian sign language recognition.

Why Cross-Validation?

In sign language recognition tasks, the model must be able to generalize across different users, hand orientations, and speeds of performing gestures. Simple train-test splits might not capture this variability, leading to overfitting or underperformance. Cross-validation mitigates this risk by leveraging all available data while also minimizing bias.

Advantages for WhisperHand:

Cross-validation significantly improves the generalization of the WhisperHand model by preventing it from overfitting to specific users or gesture variations. Since sign language gestures can vary significantly between users and repetitions, this approach ensures that the model learns patterns that are consistent across diverse scenarios. Moreover, cross-validation makes efficient use of the collected data by utilizing all repetitions during both training and validation, thereby maximizing data usage and reducing the chances of bias. By continuously testing the model with different data subsets, we ensure that the model remains robust and adaptable to new gestures and users, minimizing the risk of overfitting.

5.2.3 Results for LSTM model training

This section will show and explain the results of LSTM model training for each frequency.

5.2.3.1 Results for LSTM model training for 44Hz

The confusion matrix, as shown in Figure 5.1, Shows that the LSTM model effectively classifies 10 sign language gestures with 44 Hz frequency. as shown by the dominant diagonal line in the confusion matrix. It accurately predicts all instances (30 out of 30) for gestures like NONE, adhaktani, akol, ashrab, good job, i am, and love. However, there are minor misclassifications: 3 instances of ”shokran” are predicted as ”moafeq”, and 1 instance of ”marhaba” is also mistaken as ”moafeq”. The moafeq class itself shows high accuracy with 33 out of 34 correct predictions. These errors may result from gesture similarities or slight execution variations. Overall, the model performs well, but reducing confusion between similar gestures could enhance accuracy.

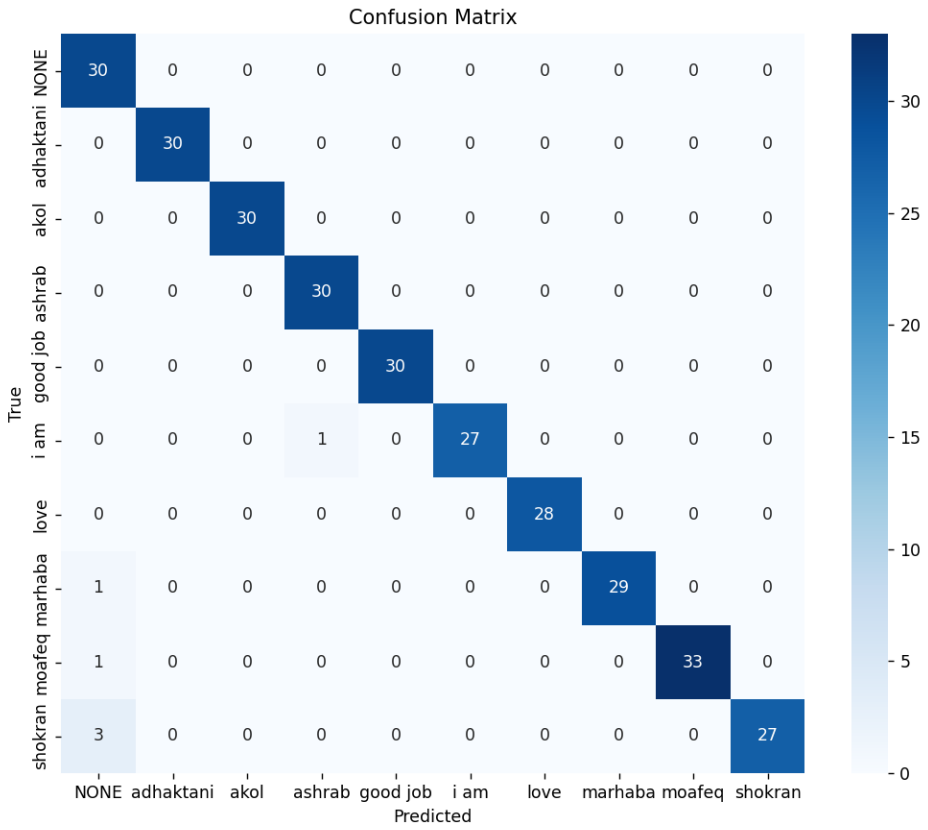

Figure 5.1: LSTM Confusion Matrix for 44Hz

Figure 5.2 depicts the training and validation performance of the LSTM model over 30 epochs. The loss curve shows a rapid decrease in training loss, stabilizing at a low value, while the validation loss initially fluctuates before converging. The accuracy curve indicates a quick rise to near-perfect

training accuracy, with validation accuracy following a similar trend despite some early instability. This suggests that the model generalizes well after initial adjustments, though early fluctuations indicate potential overfitting that could be addressed with further tuning.

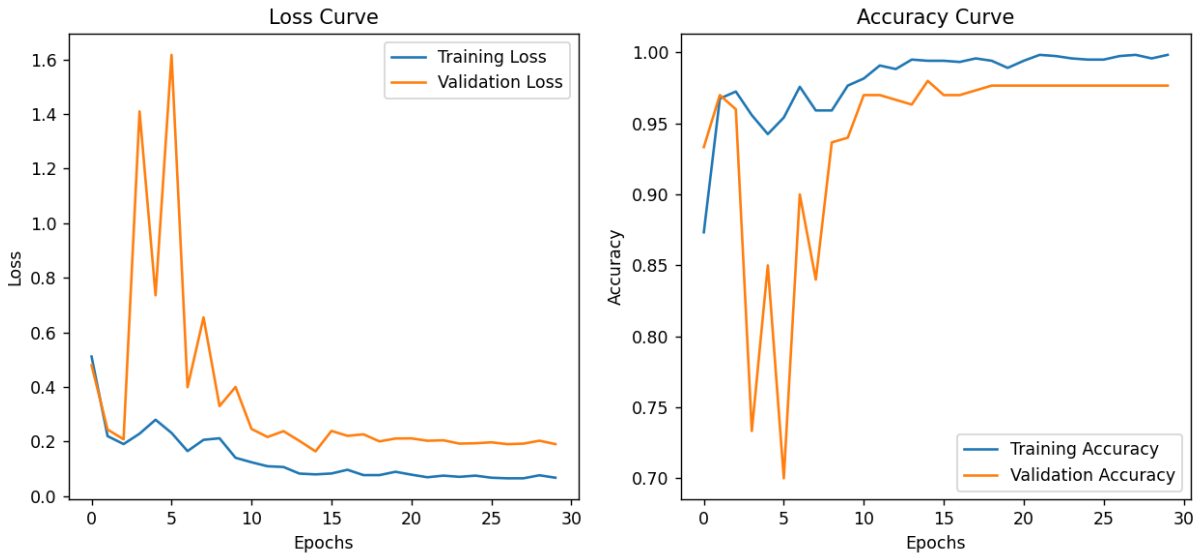

Figure 5.2: LSTM Accuracy & Loss Curves for 44Hz

Table 5.1 shows the per-class accuracy of the LSTM model for sign language gesture recognition, grouped based on accuracy levels. The model achieves perfect accuracy (1.00) for most classes, including ”NONE”, ”adhaktani”, ”akol”, ”ashrab”, ”i am”, and ”love”, indicating high consistency in recognizing these gestures. In contrast, the model shows slightly lower accuracy for ”marhaba” (0.97), ”moafeq” (0.97), ”good job” (0.96), and ”shokran” (0.90). The support column indicates the number of instances for each class, with values ranging from 28 to 34. These results suggest that while the model performs well overall, gestures like ”shokran” and ”good job” may benefit from further fine-tuning to improve classification accuracy.

Table 5.1: Per-Class Accuracy of LSTM Model for 44Hz (Grouped by Accuracy Level)

Class	Per-Class Accuracy	Support
NONE	1.00	30
adhaktani	1.00	30
akol	1.00	30
ashrab	1.00	30
good job	1.00	30
love	1.00	28
marhaba	1.00	29
moafeq	0.971	34
i am	0.964	28
shokran	0.90	30

5.2.3.2 Results for LSTM model training for 83Hz

The figure 5.3 shows the progression of accuracy and loss during training and validation, indicating stabilization of accuracy and a decrease in loss as the number of epochs increases.

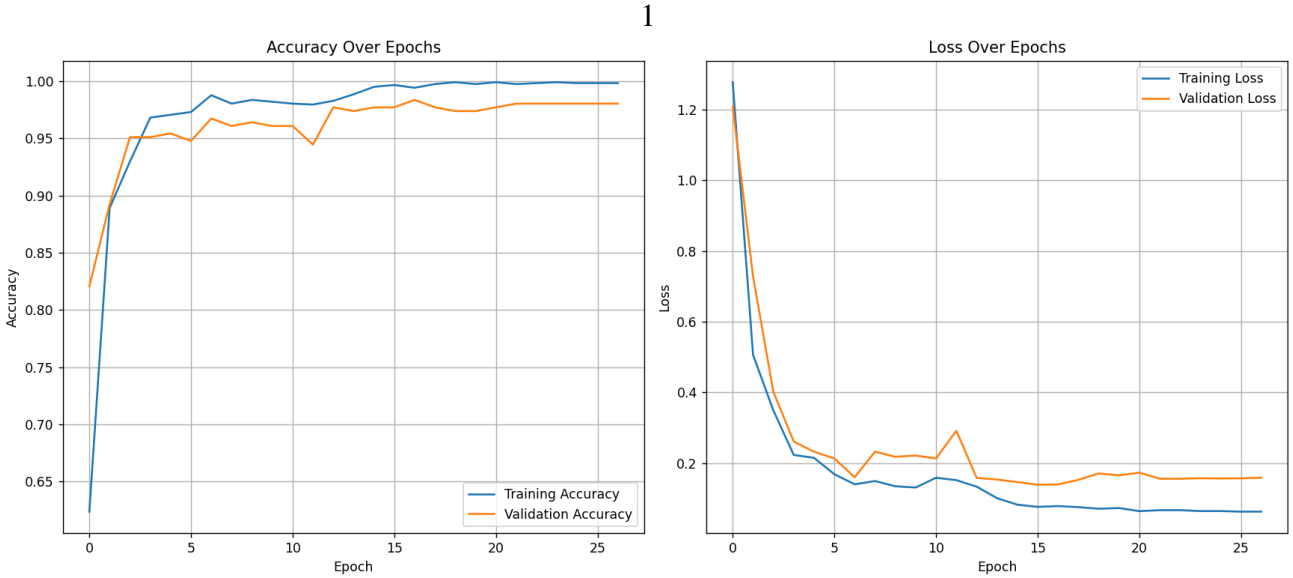

Figure 5.3: Training and Validation Accuracy and Loss Curves of the LSTM Model for 83Hz

For the confusion matrix, shown in Figure 5.4, shows the diagonal dominance in Figure 5.4indicates high accuracy, with most predictions correctly matching the true labels, while a few off-diagonal elements reflect minor misclassifications.

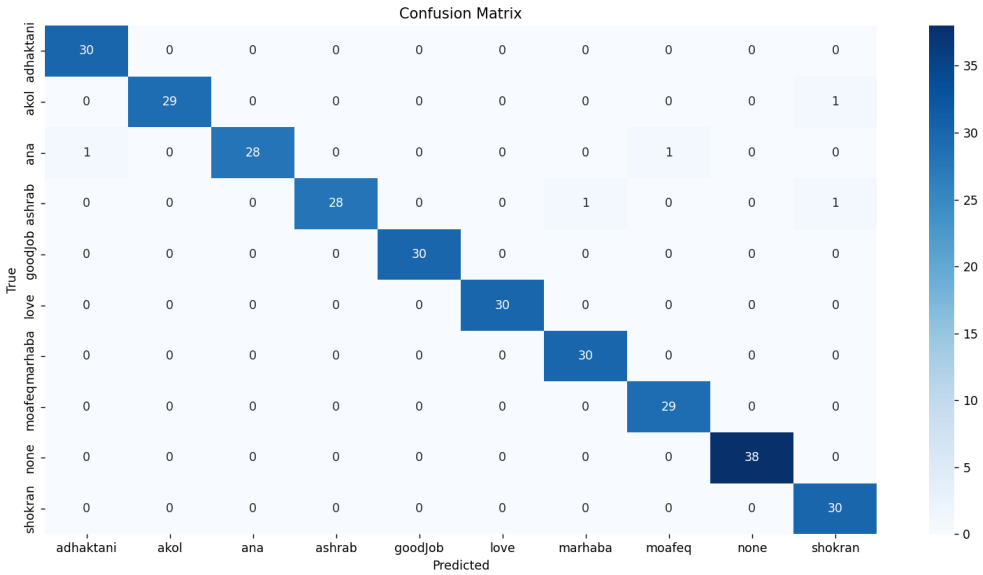

Figure 5.4: Confusion Matrix for 83Hz

Table 5.2 presents the accuracy of the LSTM model at 83Hz for each Palestinian sign language gesture, along with the number of samples (support) per class.

Table 5.2: Per-Class Accuracy and Support for Each Gesture (83Hz)

Class	Per-Class Accuracy	Support
adhaktani	1.00	30
good job	1.00	30
love	1.00	30
marhaba	1.00	30
moafeq	1.00	29
none	1.00	38
shokran	1.00	30
akol	0.967	30
ana	0.933	30
ashrab	0.933	30

5.2.3.3 Results for LSTM model training for 250Hz

The confusion matrix, shown in Figure 5.5, shows the diagonal dominance in the figure indicates high accuracy, with most predictions correctly matching the true labels, while a few off-diagonal elements reflect minor misclassifications.

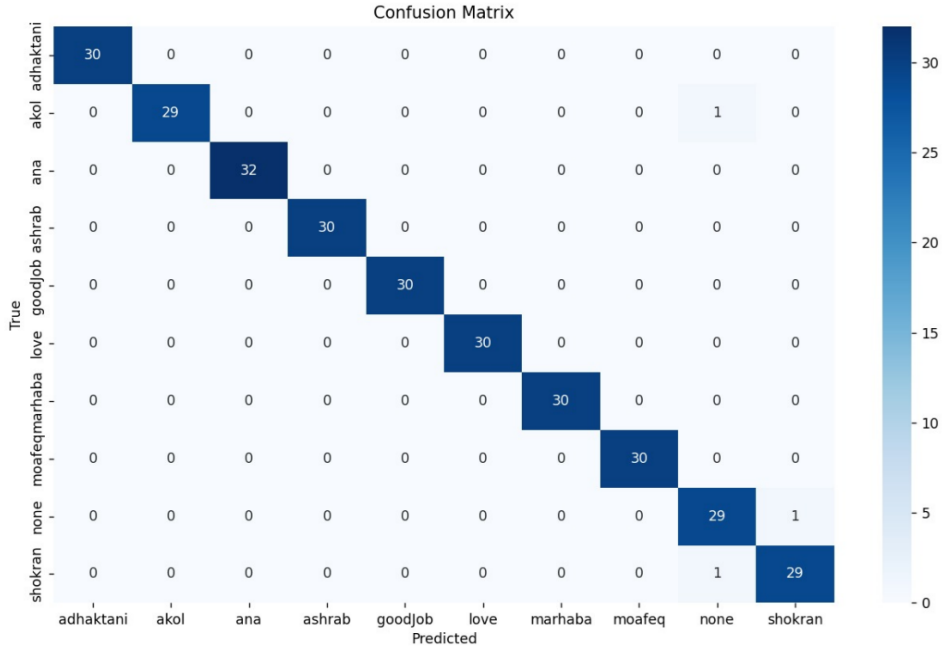

Figure 5.5: Confusion Matrix for 250Hz

Table 5.3 displays the precision and support values for each palestinian sign language gesture classified by the LSTM model at 250Hz.

Table 5.3: Classification Report: Precision and Support for Each Gesture

Class	Precision	Support
adhaktani	1.00	30
akol	1.00	30
ana	1.00	32
ashrab	1.00	30
goodJob	1.00	30
love	1.00	30
marhaba	1.00	30
moafeq	1.00	30
none	0.94	30
shokran	0.97	30

The plots in Figure 5.6 show accuracy and loss over 60 epochs at 250Hz. Training accuracy (blue) and validation accuracy (green) quickly converge to nearly 100%. Training loss (red) decreases rapidly and stabilizes, while validation loss (orange) spikes initially before stabilizing, indicating high model accuracy with minimal overfitting.

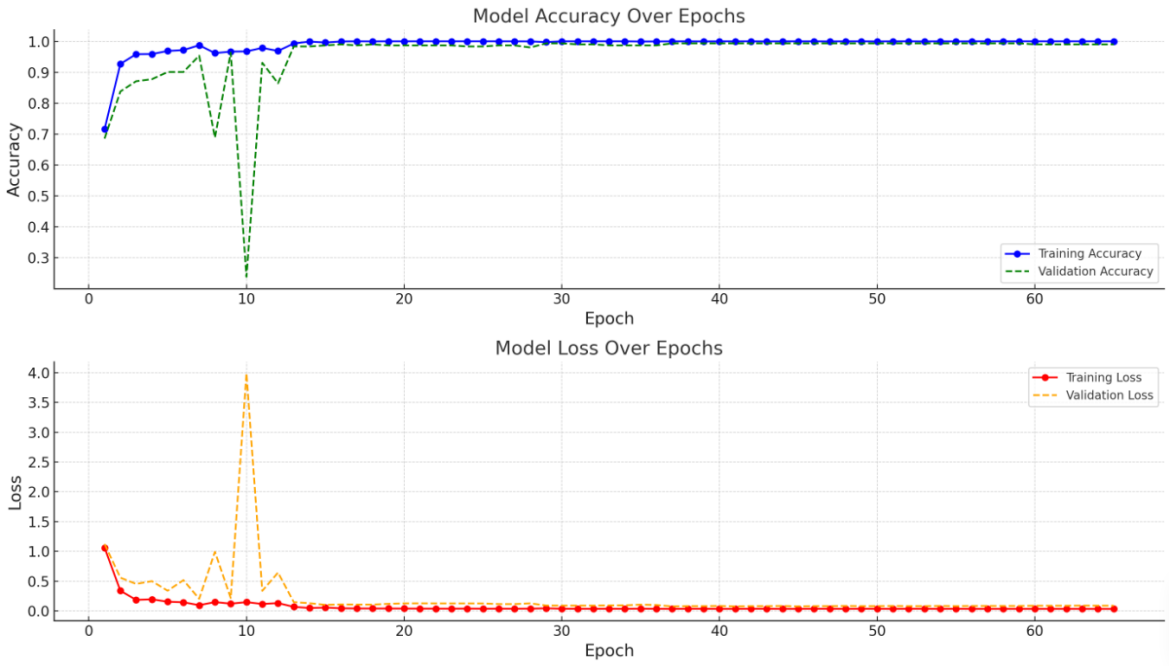

Figure 5.6: Training and Validation Accuracy and Loss Curves

5.2.4 Results for CONV1D model training

This section will show and explain the results of conv1D model training for each frequency.

5.2.4.1 Results for CONV1D model training for 44Hz

The confusion matrix shown in Figure 5.7 shows the performance of the Conv1D model in classifying ten sign language gestures. The model achieves perfect accuracy for most classes, as evidenced by the diagonal line indicating true and predicted labels match. It correctly classifies all instances (30 out of 30) for gestures like ”NONE,” ”adhaktani,” ”akol,” ”ashrab,” ”good job,” ”love,” ”marhaba,” ”moafeq,” and ”shokran.”

A slight error occurs in the ”I am” class, where one instance is misclassified as ”NONE,” resulting in an accuracy of 27 out of 28 for that class. This suggests the model may confuse similar gestures.

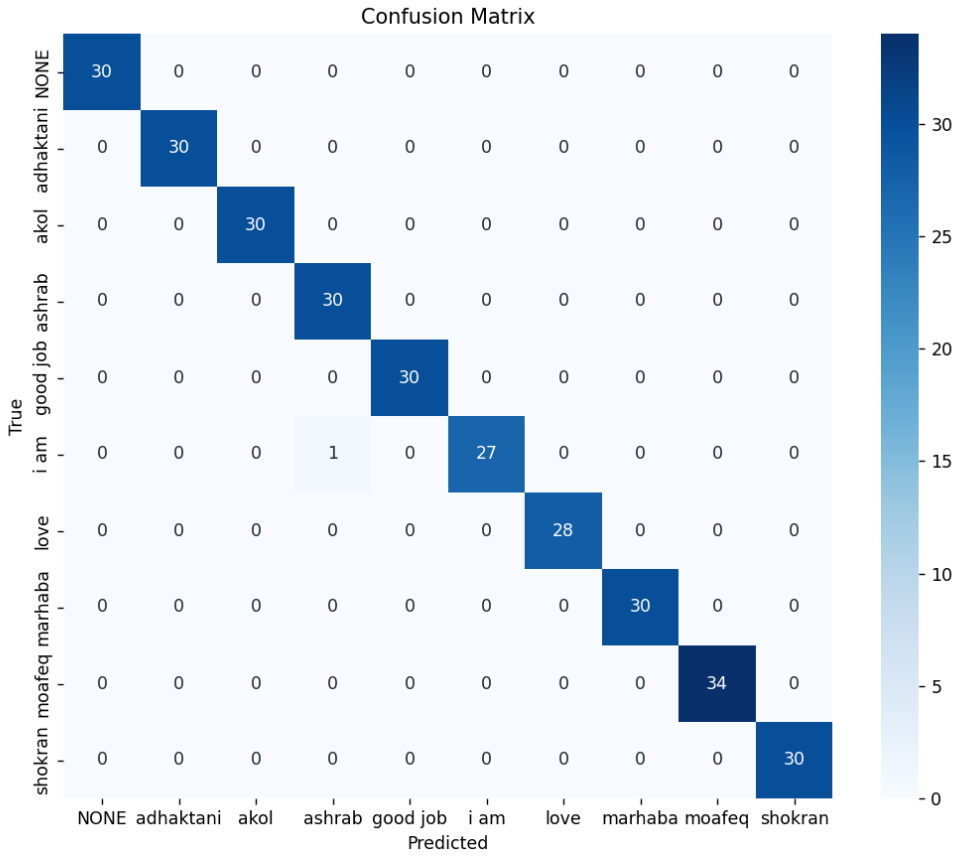

Figure 5.7: Confusion Matrix for CONV1D

Curves in Figure 5.8 shows Training and validation accuracy and loss curves for the CONV1D model over 35 epochs. The accuracy plot shows a rapid increase during the initial epochs, stabilizing around 98-99%, indicating effective learning. The loss plot demonstrates a sharp decline at the beginning, followed by a gradual convergence towards a low value, highlighting the model's ability to minimize errors and generalize well.

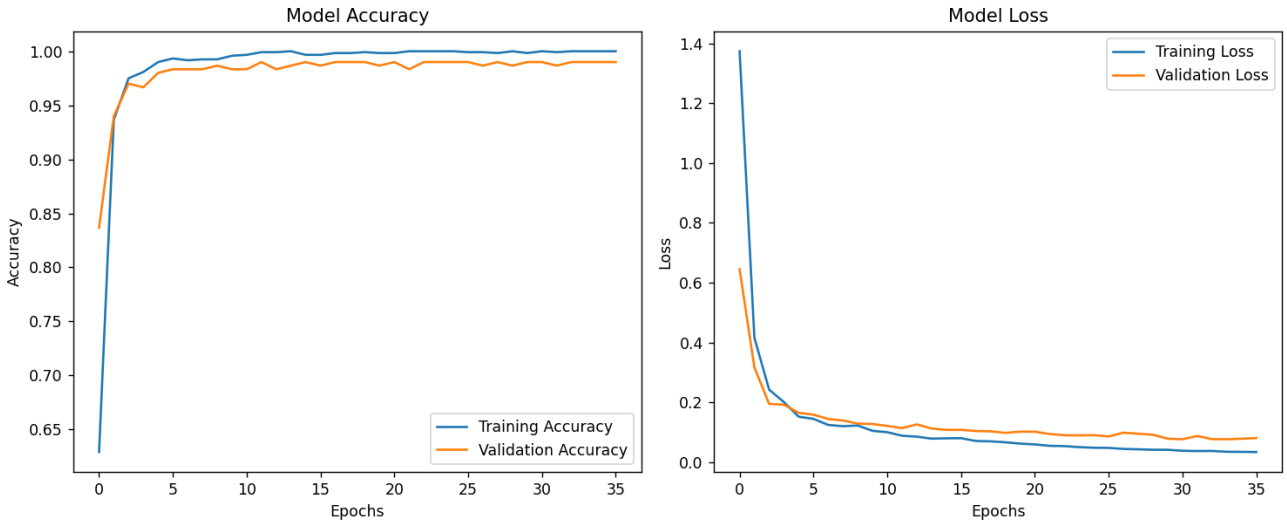

Figure 5.8: Accuracy & Loss Curves for CONV1D

Table 5.4 presents the per-class accuracy of the CONV1D model in recognizing different sign language gestures. The model achieves perfect accuracy (1.00) for most of the gestures, including ”NONE”, ”adhaktani”, ”akol”, ”ashrab”, ”good job”, ”love”, ”marhaba”, ”moafeq”, and ”shokran”. However, the gesture ”i am” shows a slightly lower accuracy of 0.96. The support column indicates the number of samples used for each class during evaluation, which ranges from 28 to 34 depending on the gesture. This high accuracy across most classes demonstrates the effectiveness of the CONV1D model in distinguishing between various sign language gestures.

Table 5.4: Per-Class Accuracy of CONV1D Model

Class	**Per-Class Accuracy**	**Support**
NONE	1.00	30
adhaktani	1.00	30
akol	1.00	30
ashrab	1.00	30
good job	1.00	30
love	1.00	28
marhaba	1.00	30
moafeq	1.00	34
shokran	1.00	30
i am	0.96	28

5.2.4.2 Results for CONV1D model training for 83Hz

The plots shown in Figure 5.9, introduces the accuracy and loss over 20 epochs. Both training and validation curves converge, indicating high performance and minimal overfitting.

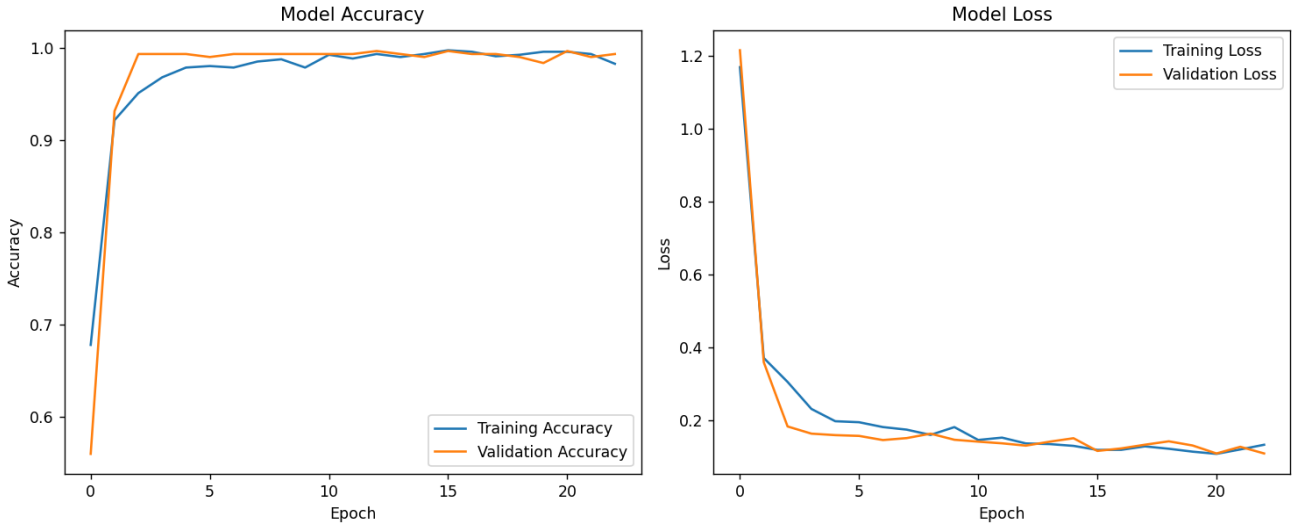

Figure 5.9: Training and Validation Accuracy and Loss in conv1D at 83Hz

The confusion matrix, shown in Figure 5.10, demonstrates high accuracy for the Conv1D model at 83Hz, with most predictions correctly on the diagonal.

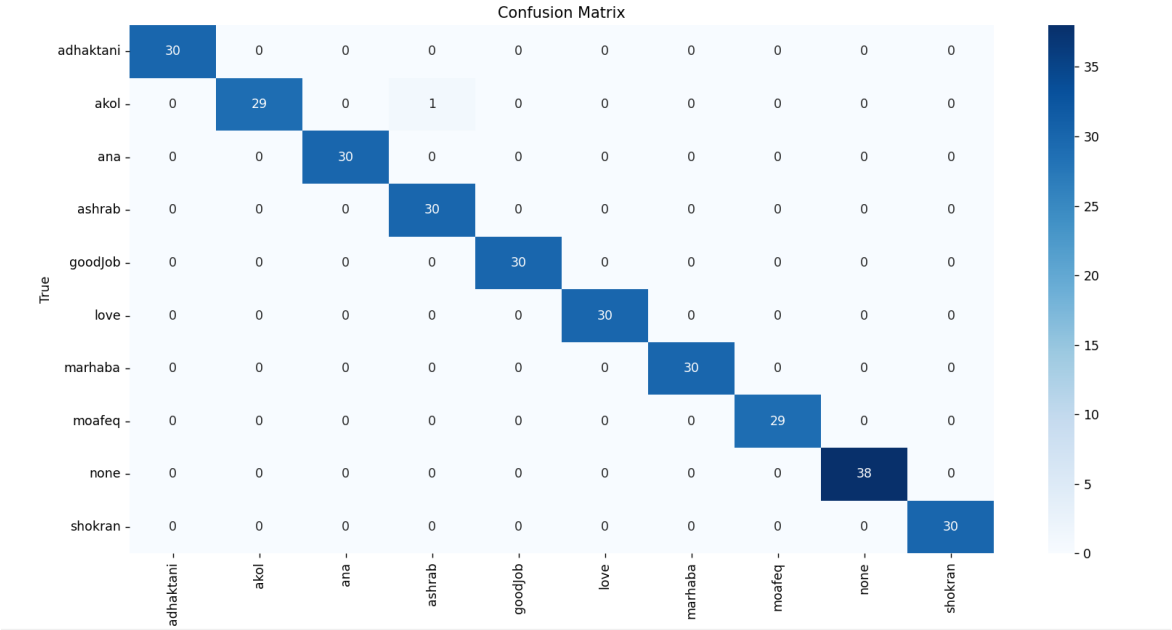

Figure 5.10: confusion matrix for conv1D at 83Hz

Table 5.5 shows per-class accuracy for the Conv1D model at 83Hz, The support values indicate the number of instances per gesture.

Table 5.5: Per-Class Accuracy and Support for Each Gesture for conv1D model at 83Hz

Class	Per-Class Accuracy	Support
NONE	100.00%	38
adhaktani	100.00%	30
akol	96.67%	30
ana	100.00%	30
ashrab	100.00%	30
goodJob	100.00%	30
love	100.00%	30
marhaba	100.00%	30
moafeq	100.00%	29
shokran	100.00%	30

5.2.4.3 Results for CONV1D model training for 250Hz

Figure 5.11 shows The diagonal dominance in the matrix shown in Figure 5.11 indicates high accuracy, with most predictions correctly matching the true labels.

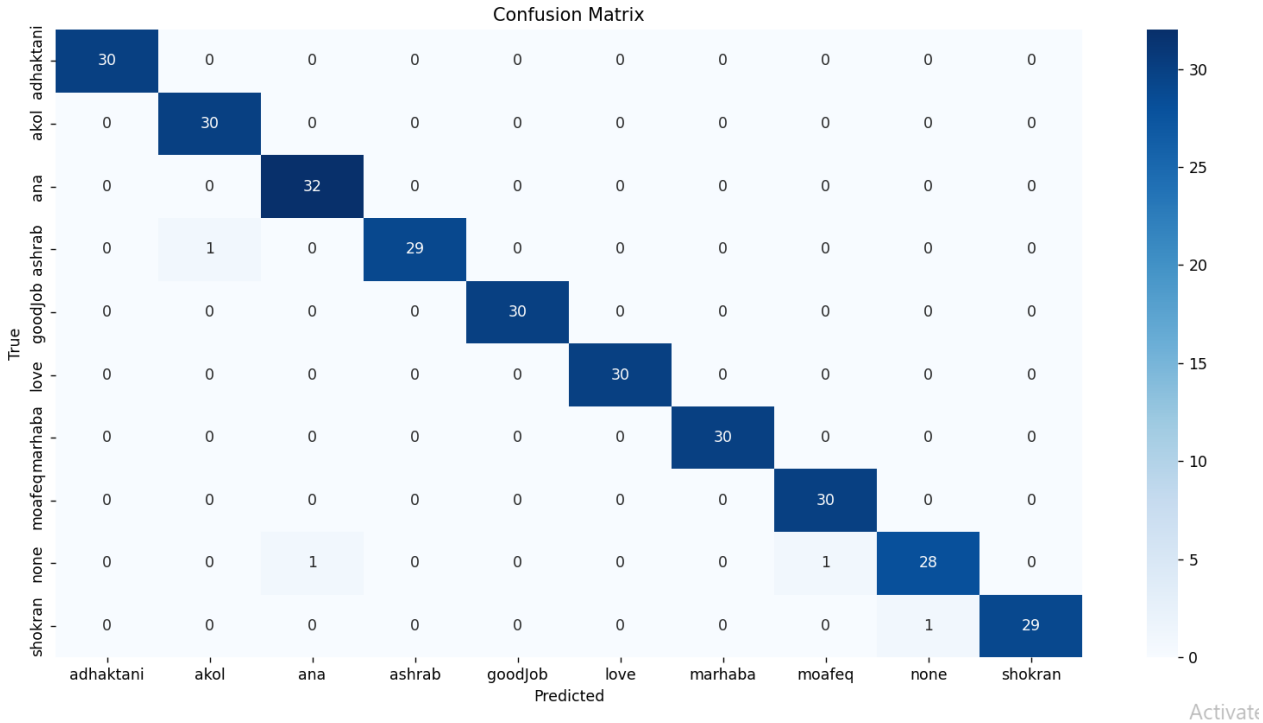

Figure 5.11: Confusion matrix for Conv1D model at 250Hz.

Figure 5.12 shows the training and validation accuracy increasing and stabilizing around 98% , while the loss decreases significantly and converges.

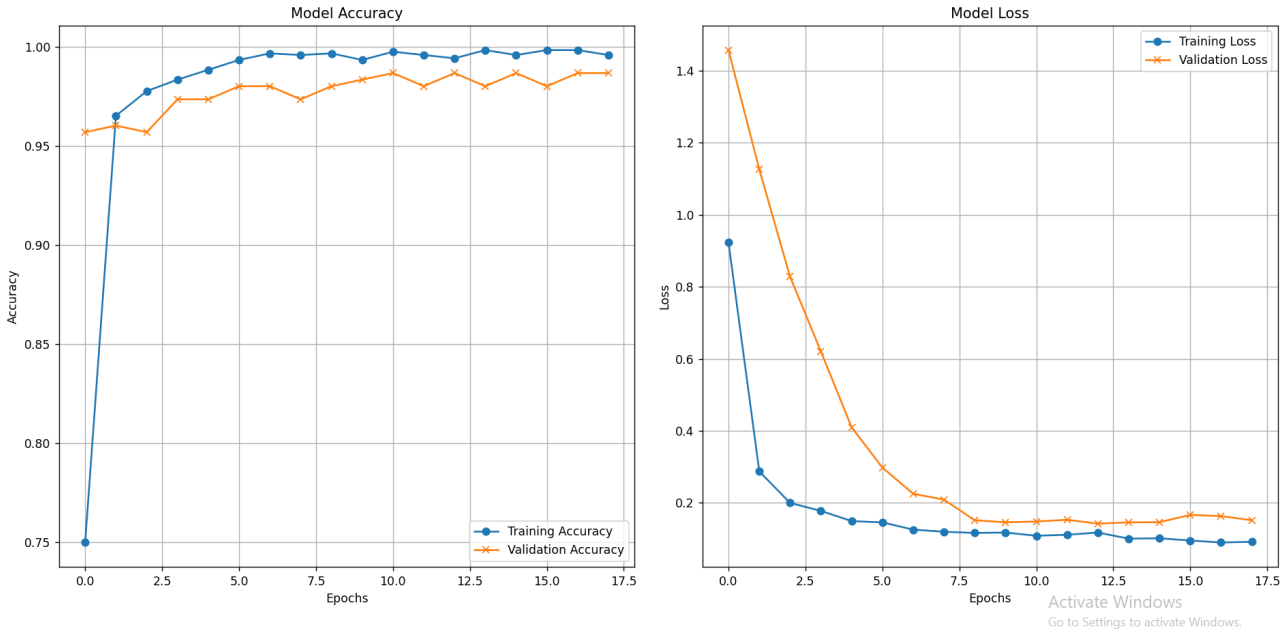

Figure 5.12: Training and validation accuracy and loss curves for the Conv1D model at 250Hz.

Table 5.6 shows per-class accuracy for the Conv1D model at 250Hz,The support values indicate the number of instances per gesture.

Table 5.6: Per-Class Accuracy and Support for conv1D model at 250Hz

Class	**Accuracy (%)**	**Support**
adhaktani	100.00	30
akol	100.00	30
ana	100.00	32
ashrab	96.67	30
goodJob	100.00	30
love	100.00	30
marhaba	100.00	30
moafeq	100.00	30
none	93.33	30
shokran	96.67	30

5.3 Real-Time Classification Performance Calculations

Table 5.7 presents the classification accuracy of the Conv1D model when tested at a sampling frequency of 44Hz. Each gesture class was evaluated under two testing conditions: Correct Order (where the sample gesture follows the same temporal pattern as training) and Reverse Order (where the gesture is played backward). The results shows a total accuracy of 89.64% in correct order where it reached 41.4% in reverse order. The notable drop in accuracy for the reverse order gestures can be attributed to the temporal sensitivity of the Conv1D model. Since the model was trained solely on forward-sequenced gestures, reversing the input sequence disrupts the learned temporal patterns, leading to significant misclassification, particularly for gestures with dynamic and time-dependent transitions.

Table 5.7: Accuracy Comparison for Conv1D Model at 44Hz (Live)

Class	**Testing Case**	**Accuracy (%)**
I am	Correct Order	99.90
	Reverse Order	33,75
marhaba	Correct Order	95.14
	Reverse Order	77.80
love	Correct Order	51.29
	Reverse Order	44.80
moafeq	Correct Order	97.70
	Reverse Order	21.66
adhaktani	Correct Order	99.45
	Reverse Order	55.51
shokran	Correct Order	99.19
	Reverse Order	60.01
ashrab	Correct Order	66.82
	Reverse Order	40.88
akol	Correct Order	97.69
	Reverse Order	15.55
Good job	Correct Order	99.60
	Reverse Order	22.99

Table 5.8 summarizes the accuracy of the Conv1D model at a sampling frequency of 250Hz, using the same evaluation criteria as the 44Hz table. The model performs exceptionally well on correctly ordered gestures, achieving accuracies up to 97.4%. However, a significant drop in performance is observed for reverse-order gestures, with some classes falling below 20%. This decline is more pro-

nounced than that observed at 44Hz, suggesting that while higher sampling frequencies enhance the model's ability to capture fine-grained temporal features, they may also increase its sensitivity to gesture directionality. As a result, reversed sequences deviate more drastically from the patterns learned during training, leading to poorer classification performance.

Table 5.8: Accuracy Comparison for Conv1D Model at 250Hz (Live)

Class	**Testing Case**	**Accuracy (%)**
I am	Correct Order	99.07
	Reverse Order	3.8
marhaba	Correct Order	98.8
	Reverse Order	14
love	Correct Order	92.7
	Reverse Order	11.18
moafeq	Correct Order	98.7
	Reverse Order	46
adhaktani	Correct Order	98.8
	Reverse Order	0.09
shokran	Correct Order	98.4
	Reverse Order	15.69
ashrab	Correct Order	91.6
	Reverse Order	24.4
akol	Correct Order	99.76
	Reverse Order	35.78
Good job	Correct Order	98.83
	Reverse Order	11.12

Table 5.9 presents the accuracy results of the LSTM model at a sampling rate of 44Hz, tested in both correct and reverse order scenarios. The model achieves near-perfect accuracy for most classes in the correct order, with values exceeding 98% in almost all cases. In the reverse order, it achieved a total accuracy of 52.7%, which is significantly higher than the Conv1D model under the same conditions. This improvement can be attributed to the LSTM's inherent ability to capture long-term dependencies and temporal patterns in sequential data. While reversing the order disrupts the gesture's original temporal flow, the LSTM model retains a level of robustness due to its memory mechanism, which allows it to generalize better across varied temporal sequences. Nonetheless, some gesture classes still showed low reverse-order accuracy—such as adhkatani—indicating that the model remains sensitive to directionality in more complex or asymmetric gestures.

Table 5.9: Accuracy Comparison for LSTM Model at 44Hz (Live)

Class	**Testing Case**	**Accuracy (%)**
I am	Correct Order	99.92
	Reverse Order	92.46
marhaba	Correct Order	99.99
	Reverse Order	56.90
love	Correct Order	89.1
	Reverse Order	38
moafeq	Correct Order	99.38
	Reverse Order	68.16
adhaktani	Correct Order	99.21
	Reverse Order	15.29
shokran	Correct Order	99.90
	Reverse Order	33.1
ashrab	Correct Order	98.68
	Reverse Order	53.2
akol	Correct Order	98.78
	Reverse Order	58.8
Good job	Correct Order	99.56
	Reverse Order	59

Table 5.10 shows the live classification accuracy of the LSTM model at 250Hz, evaluated on gestures performed in both the correct and reverse temporal order. While the model demonstrates exceptionally high accuracy (around 99%) for all gestures in the correct order, the accuracy drastically drops in reverse-order testing, reaching 0% for most gestures and staying below 7% for all. This sharp decline indicates the LSTM model's strong dependence on the temporal direction of the input sequence, which is expected given the model's architecture. The degradation is further amplified at higher sampling rates, as 250Hz captures more fine-grained temporal details, making the reversed input sequences significantly different from the original training patterns. Consequently, the temporal structure learned during training is distorted when gestures are played backward, causing the model to misclassify them. These results emphasize the importance of maintaining temporal consistency when deploying LSTM-based models in real-time gesture recognition systems, particularly at higher sampling frequencies.

Table 5.10: Accuracy Comparison for LSTM Model at 250Hz (Live)

Class	Testing Case	Accuracy (%)
I am	Correct Order	99.771
	Reverse Order	0.11
marhaba	Correct Order	99.865
	Reverse Order	0.00
love	Correct Order	99.67
	Reverse Order	20.00
moafeq	Correct Order	99.91
	Reverse Order	0.00
adhaktani	Correct Order	92.04
	Reverse Order	0.00
shokran	Correct Order	98.3
	Reverse Order	0.00
ashrab	Correct Order	99.00
	Reverse Order	7.00
akol	Correct Order	99.6
	Reverse Order	5.1
Good job	Correct Order	99.89
	Reverse Order	0.164

5.4 Challenges in Testing

Throughout the testing phase of the WhisperHand system, several challenges were encountered due to the nature of the hardware, gesture variability, and real-time constraints. These factors directly affected the reliability and accuracy of the model, especially during live classification.

5.4.1 Sensor Noise and Signal Instability

The flex sensors and MPU6050 used in the glove occasionally produced noisy or unstable readings, particularly due to:

1. Sudden hand movements
2. Loose connections or sensor drift
3. Interference from environmental vibrations

This noise led to fluctuations in the input signals, which affected the consistency of gesture classification especially in real-time scenarios. Although filtering techniques were applied (e.g. low-pass filters), complete stability wasn't always guaranteed.

5.4.2 Variability in Gesture Execution

Despite training on multiple repetitions of each gesture, there were noticeable differences in:

1. Gesture speed
2. Hand angle or positioning
3. Pressure applied to the sensors

Such variations, especially when performed by different users or at different times, led to occasional misclassifications. This revealed the model's sensitivity to gesture inconsistency even within the same label class.

5.4.3 Real Time Performance Deviation

The model performed slightly better during offline (pre-recorded) evaluation compared to live usage. In real-time testing:

1. Sensor data arrived with slight delays due to serial communication latency.
2. The model sometimes received incomplete or out of synchronization in time sequences.
3. Quick transitions between gestures caused occasional overlap or incorrect predictions.

These issues highlighted the gap between controlled testing environments and real deployment scenarios.

5.5 Result Analysis

This section presents a detailed analysis of the real time testing results to ensure that the system meets its primary objective accurate, and achives a reliable performance in real time scenarios.

5.5.1 General Analysis

1. Conv1D Model

    - At 44 Hz: The Conv1D model demonstrates a strong classification performance in the Correct Order scenario, with accuracy up to 89.64%. However, when it is tested under the Reverse Order condition, where the temporal direction of the gesture sequence is inverted, The accuracy drops moderately, ranging between 25% and 70%. This demonstrates that while the model is relatively robust, it exhibits a moderate sensitivity to the temporal ordering of input sequences.
    - At 250 Hz: The model achieves exceptional accuracy in the Correct Order case, with some classes exceeding 97.4%. Nonetheless, the Reverse Order accuracy significantly deteriorates, with several gesture classes dropping below 20%. This indicates that increasing the sampling frequency amplifies the model's reliance on gesture directionality and temporal consistency.

2. LSTM Model

    - At 44 Hz: The LSTM model achieves near-perfect accuracy in Correct Order evaluations, with total accuracy surpassing 98%. In contrast, there is a noticeable decline in performance under Reverse Order conditions, with accuracy dropping below 52.7% for certain gestures. This reflects the LSTM's inherent reliance on temporal dependencies due to its sequential learning structure.
    - At 250 Hz: While maintaining excellent performance in Correct Order scenarios with total accuracy of 98.67 %, the model exhibits a near-total collapse in Reverse Order accuracy

dropping to below 7% for the majority of gesture classes. This dramatic decline confirms the model's strong dependence on the correct temporal sequencing of input data, which becomes even more pronounced at higher sampling frequencies.

Table 5.11 summarizes the key differences between the Conv1D and LSTM models under varying sampling frequencies and test conditions:

Table 5.11: Comparison of Model Accuracy Under Different Sampling Frequencies and Temporal Conditions

Model	**Frequency**	**Correct Order Accuracy**	**Reverse Order Accuracy**	**Remarks**
Conv1D	44 Hz	Excellent	Moderate	Moderately affected by sequence order
Conv1D	250 Hz	Very High	Low	Highly sensitive to temporal order at high frequency
LSTM	44 Hz	Near-Perfect	Low	Strong dependence on motion sequence
LSTM	250 Hz	Ideal	Nearly Absent	Extremely sensitive to sequence direction

5.5.2 Gesture-Level Insight

The results indicate that the gestures "adhaktani" and "shokran" consistently showed the poorest performance in the Reverse Order scenario, suggesting that their motion patterns contain distinctive start or end points that the models struggle to interpret when the temporal sequence is reversed. In contrast, gestures like "moafq" and "marhaba" consistently performed well in the Correct Order across all experiments but experienced a sharp drop in accuracy under Reverse Order testing, This highlights the models' strong reliance on the original temporal structure of these gestures.

5.5.3 Final Observations

The Conv1D model demonstrated relatively strong resilience when handling reversed gesture sequences compared to the LSTM model. While LSTM achieved highly accurate results in the Correct Order scenarios, it showed a heavy reliance on the sequential structure of the data, leading to significant performance drops in Reverse Order tests. Additionally, increasing the sampling frequency to 250

Hz improved the models' accuracy in correctly ordered sequences but also made both models more sensitive to the temporal direction of the input, emphasizing the importance of consistent gesture flow in real-time applications.

5.6 Summary

This chapter outlines the testing and validation strategies used to evaluate the WhisperHand system. It begins with data splitting and the use of cross-validation to ensure the generalizability of models across diverse gesture executions. The chapter then presents detailed results for both LSTM and Conv1D models across different sampling frequencies (44Hz, and 250Hz), highlighting strong classification performance, especially in offline scenarios. It also includes real-time testing results, showing a drop in accuracy for reversed gestures and live classification due to gesture variability, sensor noise, and timing delays. Finally, the chapter discusses key testing challenges and emphasizes the system's robustness while recognizing areas for improvement in real-time deployment.

Chapter 6

Conclusion and future work

6.1 Concluding Remarks

Summary of Achievements

The WhisperHand system has successfully achieved its main goal of enabling real-time translation of Palestinian sign language gestures into audible speech. It integrates a wearable glove equipped with flex sensors and a motion sensor, utilizing deep learning techniques such as LSTM and Conv1D to ensure high accuracy in gesture classification. Additionally, the system is designed to be low-cost, portable, and adaptable for everyday use.

Performance Evaluation

The models were tested extensively using different sampling frequencies (44Hz and 250Hz), achieving over 95% accuracy in most cases during offline evaluations. Real-time testing also produced promising results, although misclassifications occasionally occurred due to reverse gesture sequences and noisy signals. These findings demonstrate the effectiveness of the selected architectures while highlighting their sensitivity to the order of gestures and consistency in gestures.

Limitations

Despite the system's success, several limitations were observed:

- The gesture vocabulary is limited to only single-hand, predefined gestures.

- Real-time classification is somewhat impacted by sensor noise and latency in serial communication.

6.2 Future Work

After implementing and testing the system, the following can be suggested as future work:

1. **Expanding Gesture Vocabulary**

   Future development should aim to increase the variety of supported gestures, including common phrases and complex expressions, to improve communication capabilities for deaf and mute users.

2. **Two-Hand Gesture Recognition**

   The current system only supports one hand. By extending it to recognize two-hand gestures, we can enable the recognition of more complex and natural sign language syntax, which will increase expressiveness and realism.

3. **Adaptive Learning**

   An adaptive learning feature could be introduced to accommodate different users. This would allow the system to fine-tune its model based on individual gesture styles, enhancing personalization and accuracy.

4. **Mobile and Wireless Deployment**

   Integrating the system with mobile devices and enabling wireless communication through Bluetooth or Wi-Fi will enhance portability and usability in everyday environments.

5. **Sensor Calibration and Noise Reduction**

   Implementing advanced filtering and automatic calibration algorithms will help stabilize sensor input, especially under real-time conditions. This will reduce noise and improve reliability.

6. **Natural Speech Output**

   Future versions can use text-to-speech (TTS) engines to generate dynamic spoken responses for

recognized gestures, rather than relying on static pre-recorded audio clips. This approach offers more flexibility in both speech content and tone.

Appendix A: Declaration and Documentation of AI Tools Used

Students Name:	Seba Atawneh, Sara Khatib, Shatha Awawda
Project Title:	Whisper Hand
Academic Supervisor Name:	Dr. Amal Al-Dweik
ID Number:	201024, 201073, 227044
Faculty / Department:	IT and Computer Engineering
Academic Year / Semester:	2024-2025

Table 6.1: AI Tools Usage in the Graduation Project

Purpose of Use	**Tool Name**	**Usage Location in Report (section/page)**	**Main Prompt Used**	**Was the Output Edited?**	**Usage %**
Text Generation	ChatGPT	Introduction, Summary	Write a technical introduction for a wearable glove that translates sign language into speech	Yes	18%
Grammar Check	Grammarly	Abstract, Background	Rephrase the following paragraph for clarity and grammar improvement	Yes	10%
Content Summarization	ChatGPT	Abstract, Conclusion	Summarize the main achievements and limitations of the project	Yes	17%
Data Analysis	ChatGPT	Testing and Validation	Result Analysis	Yes	10%
Diagram / Visualization	Not Used	-	-	No	0%

Purpose of Use	**Tool Name**	**Usage Location in Report (section/page)**	**Main Prompt Used**	**Was the Output Edited?**	**Usage %**
Programming Code Generation	ChatGPT	Code Appendix, Training Section	Edit Conv1D model for time-series gesture data	Yes	18%

Appendix B: Sign Language Gestures Used in the Project

This appendix presents a collection of Arabic sign language gestures used in the project.

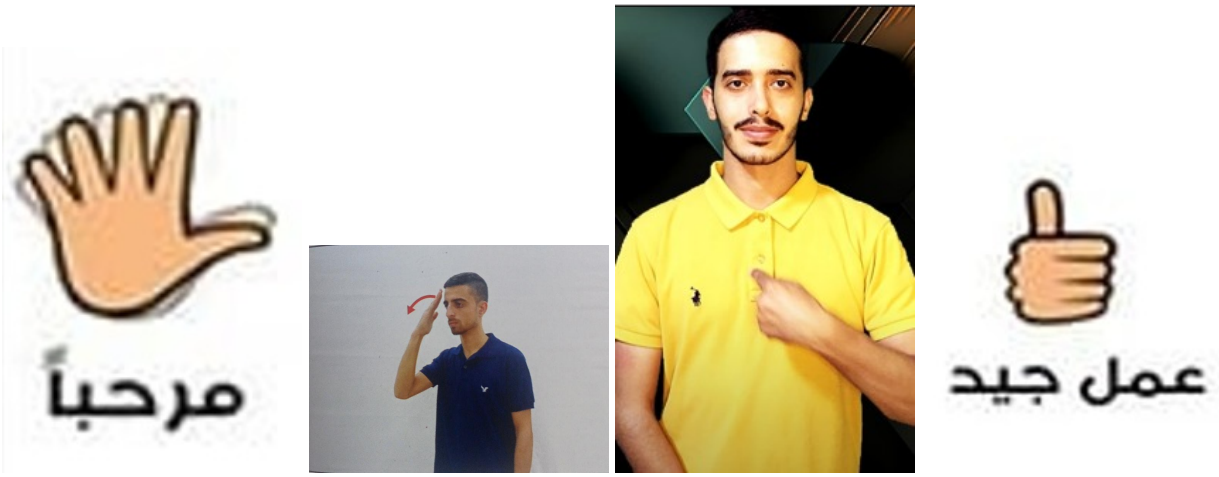

Figure 6.1: Top Row: marhaba, shokran, i am, Good Job

Figure 6.2: Middle Row: moafeq, ashrab, akol, love

Figure 6.3: Bottom Row: adhaktani

Appendix C: Source Code Snapshots

This appendix provides visual snapshots of the algorithms implemented in the project. Each image corresponds to a specific function or model used in gesture recognition.

C.1 Data Preprocessing

```python
import os
import json
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# ---------- إعداد المسارات ----------
DATASET_DIR = 'DataSet'
SAVE_DIR = 'processed_data_raw'
os.makedirs(SAVE_DIR, exist_ok=True)

# ---------- دالة قص البيانات ----------
def trim_or_pad_sequence(seq, target_length=88, num_features=11):
    if len(seq) > target_length:
        return np.array(seq[:target_length], dtype='float32')
    elif len(seq) < target_length:
        padding = np.zeros((target_length - len(seq), num_features), dtype='float32')
        return np.vstack((seq, padding))
    return np.array(seq, dtype='float32')

# ---------- دالة تحميل البيانات ----------
def load_data(dataset_dir):
    X_raw, y = [], []
    for label in sorted(os.listdir(dataset_dir)):
        label_dir = os.path.join(dataset_dir, label)
        if not os.path.isdir(label_dir):
            continue
        for file in os.listdir(label_dir):
            if file.endswith('.json'):
                file_path = os.path.join(label_dir, file)
                with open(file_path, 'r') as f:
                    data = json.load(f)
                    values = data.get('payload', {}).get('values', [])
                    if values:
                        trimmed_values = trim_or_pad_sequence(values)
                        X_raw.append(trimmed_values)
                        y.append(label)

    X_raw = np.array(X_raw)
    return X_raw, np.array(y)

# ---------- تحميل ومعالجة البيانات ----------
print("📦 Loading dataset (raw only)...")
X, y = load_data(DATASET_DIR)
print(f"✅ Data loaded: {X.shape}, Labels: {len(y)}")

# ---------- تقسيم البيانات ----------
print("🔄 Splitting into train and test sets...")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ---------- ترميز التصنيفات ----------
print("🔤 Encoding labels...")
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

y_train_onehot = np.eye(len(label_encoder.classes_))[y_train_encoded]
y_test_onehot = np.eye(len(label_encoder.classes_))[y_test_encoded]

# ---------- حفظ البيانات ----------
print("💾 Saving processed raw data...")
np.save(os.path.join(SAVE_DIR, 'X_train.npy'), X_train)
np.save(os.path.join(SAVE_DIR, 'y_train.npy'), y_train_onehot)
np.save(os.path.join(SAVE_DIR, 'X_test.npy'), X_test)
np.save(os.path.join(SAVE_DIR, 'y_test.npy'), y_test_onehot)
np.save(os.path.join(SAVE_DIR, 'label_names.npy'), label_encoder.classes_)

print("✅ Done! Raw data saved in:", SAVE_DIR)
```

Figure 6.4: Data Preprocessing

C.2 Conv1D Model Training Code

```python
import os
import json
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv1D, BatchNormalization, MaxPooling1D,
                                     GlobalAvgPool1D, Dense, Dropout, Reshape)
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from sklearn.metrics import classification_report

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# ---------------- Data Preparation ----------------
DATASET_DIR = 'DataSet'
SAVE_DIR = 'processed_data_raw'
os.makedirs(SAVE_DIR, exist_ok=True)

def trim_or_pad_sequence(seq, target_length=88, num_features=11):
    """Ensure all sequences have consistent length"""
    if len(seq) > target_length:
        return np.array(seq[:target_length], dtype='float32')
    elif len(seq) < target_length:
        padding = np.zeros((target_length - len(seq), num_features), dtype='float32')
        return np.vstack((seq, padding))
    return np.array(seq, dtype='float32')

def load_data(dataset_dir):
    """Load and preprocess sensor data"""
    X_raw, y = [], []
    for label in sorted(os.listdir(dataset_dir)):
        label_dir = os.path.join(dataset_dir, label)
        if not os.path.isdir(label_dir):
            continue
```

Figure 6.5: Code for Conv1D model

```python
# ----------------- Improved Conv1D Model -----------------
def build_conv1d_model(input_shape=(88, 11), num_classes=num_classes):
    model = Sequential([
        Reshape(input_shape, input_shape=input_shape),

        # Conv Block 1
        Conv1D(64, kernel_size=5, activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling1D(pool_size=2),
        Dropout(0.2),

        # Conv Block 2
        Conv1D(128, kernel_size=3, activation='relu', padding='same'),
        BatchNormalization(),

        # Conv Block 3
        Conv1D(256, kernel_size=3, activation='relu', padding='same'),
        BatchNormalization(),
        GlobalAvgPool1D(),

        # Classifier
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

model = build_conv1d_model()
model.summary()

# ----------------- Training -----------------
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6),
    ModelCheckpoint('best_conv1d_model.h5', monitor='val_accuracy', save_best_only=True)
]

print("🚀 Training model...")
history = model.fit(
    X_train, y_train_onehot,
    epochs=100,
    batch_size=32,
    validation_data=(X_test, y_test_onehot),
    callbacks=callbacks,
    verbose=1
)

# ----------------- Evaluation -----------------
model.load_weights('best_conv1d_model.h5')
test_loss, test_acc = model.evaluate(X_test, y_test_onehot, verbose=0)
print(f"\n✅ Final Test Accuracy: {test_acc*100:.2f}%")

y_pred = np.argmax(model.predict(X_test), axis=1)
print("\n📊 Classification Report:")
print(classification_report(y_test_encoded, y_pred, target_names=label_encoder.classes_))

# ----------------- Save Results -----------------
model.save('sign_language_conv1d.h5')
np.save(os.path.join(SAVE_DIR, 'label_names.npy'), label_encoder.classes_)
print("💾 Model and labels saved successfully!")
```

Figure 6.6: Conv1D Code Continue...

C.3 LSTM Model Training Code

```
# First LSTM layer
x = LSTM(512, return_sequences=True)(inputs)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

# Second LSTM layer
x = LSTM(256, return_sequences=True)(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

# Third LSTM layer
x = LSTM(128, return_sequences=True)(x)

# Attention layer
attention = Attention()([x, x])  # Using the same input as both query and value
x = tf.keras.layers.Concatenate()([x, attention])  # Concatenate attention output

x = BatchNormalization()(x)

# GRU layer to enhance long-term sequence modeling
x = GRU(64)(x)
x = LayerNormalization()(x)

# Final Dense layers
x = Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.0005))(x)
x = Dropout(0.3)(x)
outputs = Dense(num_classes, activation='softmax')(x)

# Build the model
model = Model(inputs, outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Callbacks for training
early_stop = EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True, verbose=1)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', patience=5, factor=0.5, verbose=1)
model_checkpoint = ModelCheckpoint('best_lstm_250hz_model.h5', monitor='val_accuracy', save_best_only=True, verbose=1)

# Train the model
history = model.fit(
    X, y,
    epochs=200,
    batch_size=16,
    validation_data=(X_test, y_test),
    callbacks=[early_stop, lr_reduce, model_checkpoint],
    shuffle=True,
    verbose=1
)

# Load best model weights if needed
model.load_weights('best_lstm_250hz_model.h5')
```

Figure 6.7: Code for training the LSTM model

Bibliography

[1] O. Surakhi, M. A. Zaidan, P. L. Fung, N. H. Motlagh, S. Serhan, M. AlKhanasfeh, R. M. Ghoniem, and T. Hussein, "Time-lag selection for time-series forecasting using neural network and heuristic algorithm," *Electronics*, vol. 12, no. 9, p. 2055, 2023.

[2] A. Shenfield and M. Howarth, "A novel deep learning model for the detection and identification of rolling element-bearing faults," *Applied Sciences*, vol. 10, no. 18, p. 6310, 2020.

[3] M. Abdel-Fattah, "Arabic sign language: A perspective," *Journal of Deaf Studies and Deaf Education*, vol. 10, pp. 212–221, Feb 2005.

[4] M. Abdel-Fattah and K. M. Alawnah, "Modality in palestinian sign language," *International Journal of Innovation Creativity and Change*, 2020.

[5] (2022) Tensorflow lite – real-time. Accessed April. 5, 2024. [Online]. Available: https://viso.ai/edge-ai/tensorflow-lite/#:~:text=TensorFlow%20Lite%20is%20supposed%20to,for%20inference%20and%20edge%20devices.

[6] (2025) What is edge impulse? Accessed April. 5, 2024. [Online]. Available: https://docs.edgeimpulse.com/docs/concepts/edge-ai-fundamentals/what-is-edge-impulse

[7] A. Jain. (2015, May) Understanding lstm networks. Accessed: May 1, 2025. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] ——. (2025, May) Understanding the 1d convolutional layer in deep learning. Accessed: May 1, 2025. [Online]. Available: https://medium.com/@abhishekjainindore24/understanding-the-1d-convolutional-layer-in-deep-learning-7a4cb994c981

[9] A. S. I. Al-Obaidi, R. R. O. Al-Nima, and T. Han, "Interpreting arabic sign alphabet by utilizing a glove with sensors," *International Journal of Health Sciences*, p. 15, 2022.

[10] O. M. A. M. Ismail Mohammed, Sulaiman Motaz Salah Amro, "Arsls peaking gloves," *PPU Dspace*, p. 53, Dec 2021.

[11] I. S. Samah Zaro, Lubna Hashlamoun, "Vocalizing arabic sign language," *PPU Dspace*, p. 112, Jun 2014.

[12] D. S. Hanaa Qasrawi, "Smart glove for translating arabic sign language "sgtarsl"," *PPU Dspace*, p. 5, Dec 2021.

[13] A. Hassoun and D. Tubeileh, "Gloves sign language translator," *ANNU Digital Library*, Jun 2017.

[14] (2025) Arduino comparison guide. Accessed Jan. 1, 2025. [Online]. Available: https://learn.sparkfun.com/tutorials/arduino-comparison-guide?utm_source=chatgpt.com

[15] S.-T. Han, H. Peng, Q. Sun, S. Venkatesh, K.-S. Chung, S. C. Lau, Y. Zhou, and V. Roy, "An overview of the development of flexible sensors," *Advanced Materials*, vol. 29, no. 33, p. 1700375, 2017.

[16] C. Pang, G.-Y. Lee, T.-i. Kim, S. M. Kim, H. N. Kim, S.-H. Ahn, and K.-Y. Suh, "A flexible and highly sensitive strain-gauge sensor using reversible interlocking of nanofibres," *Nature Materials*, vol. 11, no. 9, pp. 795–801, 2012.

[17] (2024) Understanding motion sensing technology. Accessed Dec. 6, 2024. [Online]. Available: https://invensense.tdk.com/products/smartindustrial/iim-42352/

[18] (2024) Accelerometer and gyroscopes sensors: Operation, sensing, and applications, analog devices. Accessed Dec. 6, 2024. [Online]. Available: https://www.analog.com/en/resources/technical-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.html

[19] (2024) Adafruit library. Accessed Dec. 6, 2024. [Online]. Available: https://docs.arduino.cc/libraries/adafruit-gfx-library/#Releases

[20] (2024) Adafruit slim speaker overview. Accessed Dec. 6, 2024. [Online]. Available: https://learn.adafruit.com/adafruit-stemma-speaker/overview

[21] J. P. Vásconez, L. I. B. López, Ángel Leonardo Valdivieso Caraguay, and M. E. Benalcázar, "Hand gesture recognition using emg-imu signals and deep q-networks," *Sensors*, vol. 22, no. 24, p. 9613, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/24/9613

[22] K. Team, "Keras callbacks documentation," 2023, accessed: 2025-05-25. [Online]. Available: https://keras.io/api/callbacks/